FOR
ATARI
COMPUTERS

LEARN TO PROGRAM FANCY FONTS

TRICKY TUTORIAL™ no.15

```
37,34:POKE 207,99:NEXT I:P
X+27,Y+88:SOUND 0,2,8,10:F
12:FOR I = LEN(ABS(INT(X/2
GOSUB 50:READ BOOK:BOOK =
= PEEK(242):LIST 800,1200
```

EDUCATIONAL SOFTWARE INC

FANCY FONTS
TRICKY TUTORIAL #15

by
RICHARD WILSON
and
STAN GILBERT

GETTING STARTED

Loading the Programs Into Memory

FANCY FONTS is written mostly in BASIC with some machine language routines for speed. This means that most of the programs can be LOADed and RUN from BASIC. Each part will tell you exactly how to use itself in the instructions. First to get started with a bang, let's run the Demo program.

CASSETTE:

Simply cue up the cassette to the beginning of the program you want (TYPEFONT.DMO). We give you the approximate number where it will be found on your tape. To find it exactly, you will have to load each program on the tape and write down the number ON YOUR RECORDER where each starts. Another way is to listen to the tape with a regular player until you hear a steady tone. That marks the beginning of the each program. When your tape is positioned, type 'CLOAD' and press RETURN. After the Atari beeps, press RETURN again (make sure the PLAY key is pressed on the recorder) and the program will start loading. After the program stops loading and the READY prompt appears on the screen again, type 'RUN' and press RETURN.

If you're using disk, the program is called 'TYPEFONT.DMO'. Just type RUN"D:TYPEFONT.DMO (real hard, huh!).

PROGRAM WON'T LOAD?

In the one in a million (hah!) chance that the programs won't load, send your defective copy to us for prompt replacement:

EDUCATIONAL SOFTWARE Inc.
4565 Cherryvale Ave.
Soquel, CA 95073

TABLE OF CONTENTS

Starts on tape at about #

INTRODUCTION 3

TYPEFONT - demo 4 0

See what you are about to learn to do. A fasinating example of using a custom font in a display.

CHARACTER EDITOR 5 95

Character sets means the way that the shapes of the keys (A, b, c, etc.) are shown on the screen. Begin here with our popular editor to create you own shapes or modify the many fonts that we include for your use.

LINKBAS .....19..... 181

This tool will take your fonts and put them into Basic programming code that anyone can use. LINKBAS and CHAREDIT are placed first on your tape or disk because you will use them often.

FONT PRINTER 28 207

If you have a printer, this tool will make editing shapes much easier by printing the shapes out for you.

FONT DEMO 30 228

A simple example to get you started.

TYPEFONT - program 31 250

The main part of the tutorial. A unique utility that accepts input as simple commands that are easy enough for anyone to use. Create your own screens using the fancy fonts that you have created.

FX-80 DOWNLOADER ..... 58 ..... disk only

If you have an Epson FX-80 printer, this tool will place your fonts into the printers memory so that anything you print will use your own designs instead of standard lettering styles.

GARDEN	285
TYPEFONT.FNT	303
COMPUTER	319
OLDE	335
STYLISH	350
GREEK	366
ANTIC4	381
MATH	395
RED	409
BLUE	424
ANIMATE	437

INTRODUCTION

Overview of the Demo Program

YE OLDE TYPEFONT uses a custom font to help you create "humanized" messages and programs, plus artistic special effects with a minimum of fuss and bother. The program includes:

A. A traditional style typeface (font).
B. A colorful set of graphics characters that are adaptable to a wide range of artistic presentations.

C. A set of easy commands to call effects such as decorative borders, scrolls, and pointing hands. Add emphasis with bird and gong sounds, blinking text and art.
D. A mini screen editor helps you justify text and messages on the screen or within decorative borders.

Though the emphasis is on fun and flamboyance, the variety of effects and moods you can create will be of interest to everyone in the family. Even the beginner can get professional results from the start. Use YE OLDE TYPEFONT as a family message center or to create video displays. You can dress up your programs, write poetry, and give your computer a "personality". YE OLDE TYPEFONT is written in BASIC and is completely compatible with any BASIC programming you may wish to add.

The Demonstration Program

When you run the demonsration program, it will present a series of displays showing you some of the things you can do with YE OLDE TYPEFONT.

A. The demonstration will run completely through without any further commands from you.
B. When the demonstration is complete, the 'READY' prompt will appear at the bottom of the screen.
C. **Try Some Experiments** This is your chance to get acquainted with some of the simple commands like GOSUB HAND, GOSUB GONG, GOSUB SCROLL, and GOSUB BIRD.
D. **BLINK!** Press the START button. Press it several times. "B" means "blink". Try these direct commands: GOSUB BON, GOSUB BOFF, and GOSUB BTIME (press the RETURN key after typing each command to make it work).
E. **GONG!** Try the direct command GOSUB GONG. Congratulations! You can now produce your own J. Arthur Rank movies!
F. **CLEAR THE SCREEN** Press the SHIFT and CLEAR keys at the same time (if you get stuck while experimenting, clear the screen and try again).

G. **HANDS UP!**

1. Try GOSUB HAND (try it several times).
2. Try GOSUB CHAND ("C" means Clear).
3. Try GOSUB HAND:GOSUB SCROLL
4. "SPECIFY U$"? What? Try U$="HELLO":GOSUB SCROLL
5. Try GOSUB CHAND:U$="HOW ABOUT A KISS?":GOS.SCROLL:GOS.BIRD
6. Try FOR N=1 TO 20:GOS.SCROLL:NEXT N
7. Try the loop again with GOSUB BALL:GOSUB BIRD
8. A list of all the GOSUB commands you can use appears in Appendix I.

H. **PRETTY BORDERS** Try this direct command:
FOR N=1 TO 50:GOSUB BORDER:GOSUB GONG:GOSUB EXIT:NEXT N

I. **GETTING STUCK** Just when you're starting to have fun, you will get stuck. That's because this is just a demonstration program. The other TYPEFONT program is for writing your own program, but who can resist trying something right away?

J. Press SYSTEM RESET and then RUN the program again. Now that you've experimented a little, you'll have a better idea of how the demonstration was built up from "GOSUB" commands.

Wasn't That Fun?

It's almost like having a new programming language at your fingertips. The demo is just one sample, but you can create many different character sets, and use each for a different purpose. Try using Ye old Typefont for business presentations, games, or adventures.

But I don't know how?

Right! First you have to learn to change the shapes of the characters. For this we provide our famous (Slightly) Character Set Editor. Some of you are moaning right now that you already have paid for the editor before in other tutorials. Some companies choose to sell editors by themselves, and never explain all the details of how to use the darn things. ESI gives the editors away for FREE, and charges for the MANY EXAMPLES and large manual explaining everything we know about the subject. This means you get our best editors free in any program where we think you need them. Now that EVERYONE is happy, here are the instructions for using the character editor:

CHARACTER EDITOR

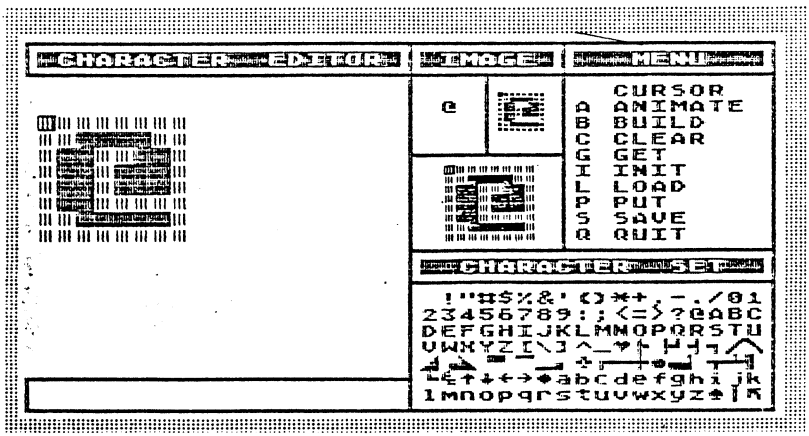
INTRODUCTION

The CHARACTER EDITOR program is named "CHAREDIT.UTL" and is run from BASIC. This utility is designed to allow you to create your own character fonts and save them. The CHARACTER EDITOR has a very nice feature called ANIMATE which allows you to construct animation incarnations and observe their animation on the screen at different speeds. You may also construct composite graphics characters in the BUILD mode. This utility uses extensive assembly language routines for the purpose of speed and you are encouraged to explore the use of assembly language USR CALLS for your own applications. They are easy to use from BASIC without knowing how they were written.

SCREEN DESCRIPTION

Once "CHAREDIT.UTL" is up and running you should become familiar with the screen layout. There are five separate major areas. They are:

1. BIT MATRIX
2. CHARACTER IMAGE DISPLAY
3. MENU
4. CHARACTER SET DISPLAY
5. USER INPUT PROMPT AREA



CHARACTER EDITOR

BIT MATRIX

This is a grid of 64 rectangles, each representing a single BIT. Each of the 8 rows of 8 BITS represents a single BYTE. They are stacked from top to bottom as they occur sequentially in the character set for each character. This is the main work area for editing, animating, or building composite characters. A composite character is explained later.

CHARACTER IMAGE DISPLAY

There are three parts to the CHARACTER IMAGE DISPLAY located to the right of the BIT MATRIX, in the upper half of the screen. These areas each contain a single representation of the character you are working on. They are in different colors for the sake of clarity and no other significance is attached to the color. The first area represents how the character will look in GRAPHICS mode zero (0). The area to the right of that represents GRAPHICS 1 and below them is GRAPHICS 2.

Since the figures representing GRAPHICS 1 and 2 occur on the same lines, you may have guessed that they are player-missiles and not actual graphics mode 1 and 2 characters. Bear in mind, the proportions will change slightly when you do use your altered character set in graphics modes 1 and 2.

MENU

This area contains the ten (10) major categories of operational functions you may perform using the CHARACTER EDITOR. As soon as any of the functions are invoked, that menu choice will be blocked in a blue stripe. This allows you to recall what you were doing when you return to the editor after some diversion. Each MENU choice will be discussed in detail in a later section.

EVERY EFFORT WAS MADE TO USE MENU FUNCTIONS UTILIZING THE FIRST LETTER OF THE DESIRED FUNCTION AS A COMMAND FOR EASY MEMORIZATION BY ASSOCIATION.

CHARACTER SET DISPLAY

The entire font you are working with is found in this display area. When you want to "GET" or "PUT" a character the cursor is placed in this area for you to locate the

desired character using the cursor controls. You may notice that ONLY 126 CHARACTERS ARE AVAILABLE FOR EDITING in the CHARACTER SET DISPLAY. That is because you are excluded from editing the BACK SPACE or DELETE characters as they are specially modified for use in the BIT MATRIX display area.

USER INPUT PROMPT AREA

The elongated rectangular area in the bottom left portion of the screen is used for error messages and user input to specific prompts. For instance, This is the area you will use to enter the filename of a FONT that you want to LOAD or SAVE.

CHARACTER EDITOR OPERATION

MANY OF THESE MENU FUNCTIONS ARE BEST ILLUSTRATED THROUGH DEMONSTRATION. WE HAVE PROVIDED YOU A FONT FOR THIS PURPOSE CALLED "GARDEN". SEE "LOAD" FOR INSTRUCTIONS IN LOADING THIS FONT FOR USE IN LEARNING HOW TO USE THE CHARACTER EDITOR.

CURSOR

The CURSOR function on the MENU indicates ANY action that manipulates the BIT MATRIX or uses the CURSOR controls while editing a character in the BIT MATRIX. There are six major CURSOR functions available:

1. MOVE CURSOR
2. WRITE CURSOR
3. HOME CURSOR
4. SHIFT MATRIX
5. BYTE NUMBERS ON/OFF
6. CURSOR ON/OFF

MOVE CURSOR

Anytime one of the four keys that have CURSOR ARROWS on them is depressed the cursor will move accordingly in the BIT MATRIX. If the cursor is at its boundary limit it will wrap around to the opposite boundary. No other action is taken other than moving the cursor to a new BIT SQUARE.

WRITE CURSOR

Holding down the CTRL (control) key while depressing one of the four keys that have CURSOR ARROWS on them will cause the cursor to TOGGLE (between ON and OFF) the BIT SQUARE it is in and then move in the appropriate direction. Movement follows the same rules as MOVE CURSOR and no action

is taken in the destination BIT SQUARE. For instance, if the cursor were sitting in a BIT SQUARE that was turned "ON" and we wanted to erase that BIT then we need only hold down the CTRL key and move the cursor out of that square. The WRITE CURSOR mode replaces the value of a BIT SQUARE ~~with~~ the opposite value as it moves from that BIT SQUARE. Remember that a bit value of 0=off and 1=ON. The result of "WRITING" will be observed in the CHARACTER IMAGE DISPLAY areas as well as in the character position in the CHARACTER SET DISPLAY AREA.

HOME CURSOR

Pressing the "H" key will place the cursor in the upper left corner of the BIT MATRIX. This position is commonly called "HOME" and that is why the "H" key is used. Holding down the CTRL key and pressing the "H" key will place the cursor in the lower right corner of the BIT MATRIX.

SHIFT MATRIX

Holding the SHIFT key down while pressing one of the four keys with CURSOR ARROWS on them will cause the entire contents of the BIT MATRIX to SHIFT one BIT ROW in the direction indicated on the cursor key. The SHIFT key is used in conjunction with the cursor keys because that is what you want to do, SHIFT the BIT MATRIX.

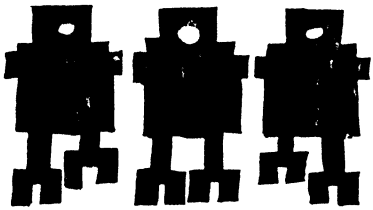
BYTE NUMBERS ON/OFF

You may want to see the value for each of the BYTES that make the BIT MATRIX. These NUMBERS may be turned on by pressing the "N" key. They will change as you WRITE different values in the BYTE that the cursor is positioned on. If you do not want to see these NUMBERS then turn them off by holding down the CTRL key while pressing "N". The CHARACTER EDITOR is initialized with these numbers turned off.

CURSOR ON/OFF

The cursor itself must be distinctly visible for your use in editing a character. You may find it distracting to observe the BIT MATRIX pattern you have created while the cursor is glaring at you. Hitting the SPACE BAR will make the cursor turn off for unobstructed viewing of the BIT MATRIX. Hitting the SPACE BAR again, or ANY OTHER CURSOR FUNCTION, will make the cursor visible.

CHARACTER EDITOR



ANIMATE

The ANIMATE function of the CHARACTER EDITOR is very unique and not found on any other character editor we know of. It actually allows you to witness the animation of a series of character incarnations, both inside the BIT MATRIX and in each of the CHARACTER IMAGE DISPLAYS. Pressing the "A" key will invoke the ANIMATION feature. You will be prompted to enter the characters you wish to observe in sequence. Once the characters are entered and you push RETURN they will begin to animate. You will notice that there is a menu prompt in the USER INPUT PROMPT AREA area that says: SLOW FAST QUIT. Entering the first letter of any of these choices will cause the animation to Slow down, go Faster, or Quit. Depending on the speed at which the animation is running and the amount of incarnations entered, there may be a delay before your request is acted upon. Nothing will happen until one entire sequence is finished. As the speed reaches its limits, that menu choice will disappear from the screen.

Using the "GARDEN" font, invoke ANIMATE and enter "#$%" and you will see PROTO stomp around. The only difficulty in using ANIMATE is remembering which characters to enter as incarnations since they can no longer be recognized as standard characters. Remember, you have just spent your valuable time creating all kinds of fancy shapes that look nothing like the keyboard. We recommend you keep a cross reference or create your incarnations bracketed between a couple of standard characters for ease of identification.

If you should invoke ANIMATE by pressing the "A" key and decide you do not want to use it, just push RETURN.

BUILD

BUILD allows you to construct a composite character that requires more than one character for definition. You invoke BUILD by pressing the "B" key. You will be prompted for the four (4) characters you wish to edit in order to build your composite. YOU MUST ENTER 4 CHARACTERS, although, you may use blanks for dummies if you only wanted to use two or three characters. After keying in your 4 characters press RETURN.

CHARACTER EDITOR

The characters you enter will appear in this order:

1 2
3 4

within the CHARACTER IMAGE DISPLAY box that represents GRAPHICS 2 mode (the largest characters) from left to right and top to bottom in the order you entered your choices. The BIT MATRIX will hold the first character you entered as a starting point. In order to GET another of the characters to work with, enter a "G" for GET. You will be prompted for a character choice and only one of the four displayed characters will be permitted. To QUIT the BUILD mode just press the "Q" key for QUIT or ANY OTHER MENU FUNCTION. Please note that you may QUIT only after you have satisfied the input requirement for four (4) characters and returned to the BIT MATRIX.

CLEAR

Pressing the "C" key for CLEAR simply clears the BIT MATRIX by setting all BITS to zero or "OFF". Whatever character is occupying the BIT MATRIX is also completely cleared in the CHARACTER SET DISPLAY AREA. This is a good way of clearing the work area if you are going to completely redefine a character and much of what is in the present character is not usable. It saves you from erasing all those BITS you're not going to use. If you CLEARed an ATARI character you may restore it while in the GET or PUT modes. See GET or PUT.

GET

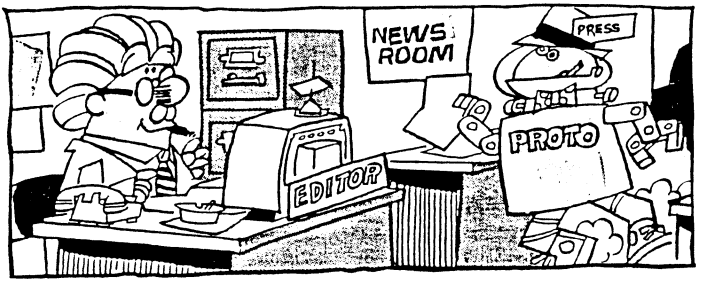
The GET command allows you to get a character from the CHARACTER SET DISPLAY AREA and put it in the BIT MATRIX for editing. You invoke GET by pushing the "G" key. The cursor will appear in the CHARACTER SET DISPLAY AREA and you can move it around using the four keys with CURSOR ARROWS on them. Once the cursor is over the character you want, just push RETURN and that character will appear in the BIT MATRIX, ready for editing. The old position in the CHARACTER SET DISPLAY AREA will be "remembered" and the cursor will return there upon the next GET or PUT operation. You may QUIT the GET function by pressing the "Q" key.

Characters may be restored to their original ATARI character while in this mode simply by placing the cursor over the desired character position and pressing the "R" key for RESTORE.

CHARACTER EDITOR

INIT

Pushing the "I" key will initialize the CHARACTER EDITOR. It is like starting all over. For safety you will be prompted "INITIALIZE (Y/N)?". ANY response other than "Y" will return you to where you left off. A "Y" RESPONSE WILL INITIALIZE THE CHARACTER EDITOR AND ANY PREVIOUS WORK WILL BE DESTROYED UNLESS IT WAS SAVED. Use this only if you have so hopelessly messed up the character set that it is easier to start over than fix the current file being edited.



LOAD

Disk Users

The LOAD function allows you to LOAD in a CHARACTER SET previously stored on tape or disk. When you press the "L" key you invoke the LOAD function and are prompted "LOAD?". The response to the prompt should be a legal filename. Extensions to the filename are permitted although NOT recommended. The default extension is ".FNT" and will be appended to the filename automatically. The default drive is "D:" and may be changed using standard filename protocols.

NORMALLY ONLY THE BODY OF THE FILENAME NEED BE ENTERED. It must be 8 uppercase characters and must start with a letter of the alphabet (ie. entering the filename "MYFONT1" would be seen as "D:MYFONT1.FNT"). If the requested filename is not accessible (say due to a bad sector or the drive or the tape not being plugged in) then a "BAD FILENAME" error message will appear in the USER INPUT PROMPT AREA. You may exit the LOAD function by pressing RETURN with nothing entered.

Tape Users

Tape users must type C:DUMMY where DUMMY is any name (it is ignored but must be there for the LOAD to work).

CHARACTER EDITOR

PUT

PUT is invoked by pressing the "P" key. Whenever you GET a character and edit it, it is NOT necessary to PUT the altered character into the same place in the character set. However, you may want to PUT the character into another location without having to edit the character in that new location.

Perhaps you are creating several incarnations of a character to be animated and only a small part of each incarnation must be re-drawn, leaving the body of each character the same. It could get tedious re-drawing each character in its entirety. When you invoke PUT you may move the cursor to any position in the CHARACTER SET DISPLAY AREA, using the four keys with CURSOR ARROWS on them, and push RETURN. This will PUT the character presently residing in the BIT MATRIX into that position of the character set. The character in the BIT MATRIX remains unchanged in the BIT MATRIX and you may PUT it as many times as you like. To QUIT the PUT mode just press the "Q" key.

You can RESTORE any character in the CHARACTER SET DISPLAY AREA to its original ATARI character while in the PUT mode by positioning the cursor over the desired character and pressing the "R" key for RESTORE.

SAVE

The SAVE function allows you to SAVE onto disk or tape the character set in the CHARACTER SET DISPLAY AREA. When you press the "S" key you invoke the SAVE function and are prompted "SAVE?". The response to the prompt should be a legal filename. Extensions to the filename are permitted although NOT recommended. The default extension is ".FNT" and will be appended to the filename automatically. The default drive is "D:" and may be changed using standard filename protocols.

NORMALLY ONLY THE BODY OF THE FILENAME NEED BE ENTERED. It must be 8 uppercase characters and must start with a letter of the alphabet (ie. entering the filename "MYFONT1" would be seen as "D:MYFONT1.FNT"). If the requested filename is not legal or any error occurs (such as DISK FULL), then a "BAD FILENAME" error message will appear in the USER INPUT PROMPT AREA. You may exit the SAVE function by pressing RETURN with nothing entered.

Tape Users

Tape users must type C:DUMMY where DUMMY is any name (it is ignored, but must be there for the LOAD to work).

CHARACTER EDITOR

USING CHARACTER SETS

You have done a lot of planning and work to create a custom character set for your special needs. Now you want to use it (how dare you!). You have this FONT stored in FILENAME.FNT and now you want to get it into your BASIC program. There are two ways to do it. They are by FILE I/O or by RESIDENT CODE.

FILE I/O

FILE I/O simply means that you are going to leave the character set in the FONT file and "INPUT" it into your program where it is needed. The following code will load your character set into RAM:

```
2000 OPEN #1,4,0,"D:FILENAME.FNT"
2010 FOR I=0 TO 1023
2020   GET #1,BYTE
2030   POKE CA+I,BYTE
2040   NEXT I
2050 CLOSE #1
```

where CA is the address for the character set. Again, cassette owners must use C: instead of D:. This method is time consuming and if you have more than one character set to load it can get annoying. We provide you with an assembly language USER CALL routine that does it very quickly. This routine is called CIOCTL and is held in the string CC$ in the CHARACTER EDITOR which you may LIST. The following code is an example of how to use CIOCTL:

```
2000 OPEN #1,4,0,"D:FILENAME.FNT"
2010 X=USR(ADR(CC$),1,7,CA,1024)
2020 CLOSE #1
```

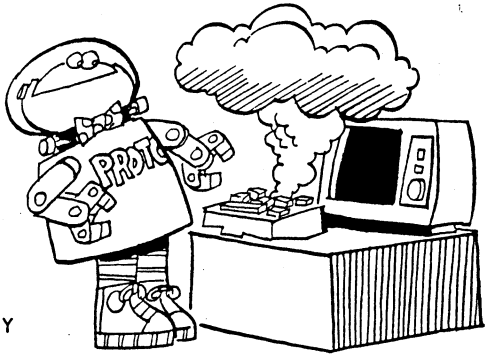
Remember, cassette users must use C: instead of D:. This is time consuming and if you have more than one character set to load it can get annoying. We provide you with an assembly language USER CALL routine that does it very quickly. This routine is called CIOCTL and is held in the string CC$ in the CHARACTER EDITOR which you may LIST. The following code is an example of how to use CIOCTL:

```
2000 OPEN #1,4,0,"D:FILENAME.FNT"
2010 X=USR(ADR(CC$),1,7,CA,1024)
2020 CLOSE #1
```

CHARACTER EDITOR

QUIT

Pressing the "Q" key will cause the prompt "QUIT (Y/N)?" to appear in the USER PROMPT AREA. Any response other than "Y" will return you to the editor where you left off. A "Y" response will restore you to BASIC.



DISASTER RECOVERY

The scenario is this: You are tired and have spent a lot of time changing several characters in your custom character set. Without thinking, you choose to QUIT the editor and the very instant you push the "Y" key to QUIT you realize you did NOT save your work! Before you commit some atrocity, be aware you CAN recover your work if you are very careful to follow these steps PRECISELY:

Remember, you are now in BASIC.
A. DO NOT TYPE IN "new"
B. DO NOT LOAD ANOTHER BASIC PROGRAM
C. DO NOT EXIT BASIC

Enter the following commands followed by RETURN

```
CLOSE #1
OPEN #1,8,0,"D:FILENAME.FNT"
X=USR(ADR(CC$),1,11,CA,1024)
```

WHAT DID I DO?

The first statement insured that channel #1 was CLOSEd. The next statement OPENed channel #1 to the filename of your choice. Finally, the last statement invoked a USER CALL within the character editor that moved 1024 BYTES of memory, starting at memory location CA (character set address), to the filename. You may now RUN the CHARACTER EDITOR normally and LOAD that filename to continue your work.

CHARACTER EDITOR

This routine can be used to quickly LOAD from or SAVE to files.

```
X=USR(ADR(CC$),CHNL,I/O,ADR,LEN)
                |    |   |    |
file channel #..|    |   |    |..# of BYTES to transfer
must match the       |   |
number used to       |   |
OPEN the file.       |   |
                     |   |
LOAD(7) or SAVE(11)..|   |..RAM address to LOAD or SAVE
```

The only problem with this technique is that extra I/O and support files provide greater opportunity for problems. The major advantage is memory conservation. If your program is reaching your memory limits then having the character sets stored on a separate file has value.

RESIDENT CODE

This means that the character set is contained within your program, either in a DATA statement a STRING variable. The following example represents having the character set in DATA statements:

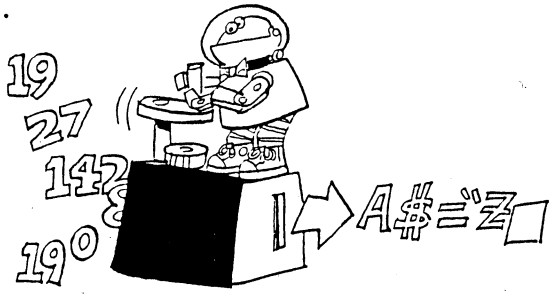
```
2000 RESTORE 7000
2010 FOR I=0 to 1023
2020 READ BYTE
2030 POKE CA + I, BYTE
2040 NEXT I

7000 DATA 0,0,112,134,34,2,2,0,0,65,65
7010 DATA (another 1013 numbers!)
```

There are two major disadvantages to this technique. First, DATA statements consume a lot memory. Why use a number like 112 which takes up 3 BYTEs when a single BYTE the value 112 would do. A lower case "p" has the ATASCII value of 112. Secondly, it is slow. If you used STRING variables to hold the data it would look like this:

```
2000 DIM CS$(1024)
2010 CS$="1024 ATASCII CHARACTERS REPRESENTING BYTE
VALUE"
2020 FOR I=0 to 1023
2030 BYTE=ASC(CS$(I+1,I+1)
2040 POKE CA+I,BYTE
2050 NEXT I
```

CHARACTER EDITOR

This is more practical from a memory conservation point of view, but, it is slow. Also, it requires some method by which the byte values can be interpreted into ATASCII characters. Well, OF COURSE! You didn't think we wouldn't provide you such a utility, did you? It is called "LINKBAS.UTL".



LINKBAS.UTL -- Disk Version

This UTILITY is designed to translate the BYTE contents of any file into BASIC STRING VARIABLES. This is useful for implementing re-locatable assembly language routines, screen formats, or character sets into BASIC programs. The operation of this utility is simple:

RUN "D:LINKBAS.UTL" from BASIC

1. Answer the first prompt with the DESTINATION filename you want to contain the converted code. Filename extensions are not required, although advised. Pressing RETURN with nothing entered will default to "D:TEMP.LST". ".LST" is is the most obvious extender as the converted code that is generated is a LIST file which must be ENTERed into a BASIC program to be used. ".ASY" might be used for converted ASSEMBLY routines. The default device "D:" may be changed by including the new device specification in the filename.

2. The next prompt is for a LINE NUMBER you want the converted code to begin with. Pushing RETURN with nothing entered will default to LINE 2000. Line numbers will increment by 10.

3. The next prompt will request a SOURCE filename and follows the same rules as the DESTINATION filename, except there is no default. However, there is a default filename extender. Pressing RETURN with a filename that has a "." on the end of it will append the default extender ".OBJ". ANY FILENAME WITH THE EXTENDER ".OBJ" IS ASSUMED TO BE ASSEMBLY LANGUAGE OBJECT CODE AND WILL IGNORE THE FIRST SIX BYTES OF THE SOURCE FILE because they are address vectors and should not be included in the string conversion. Entering the

filename "MYGAME.FNT" would put the re-defined character set found in the FONT file MYGAME into a string variable. The last prompt requires a LEGAL STRING VARIABLE NAME up to 2 characters to define the conversion. Here is the format for answering the prompt:

SOURCEFILE/VAR<@>

where the sourcefile must have a legal name with no spaces followed by a slash. You can change which drive you wish to use by attaching it to the front of the filename. The extent .OBJ will be appended to the filename. Following the first slash is the variable name of your choosing. It is highly recommended that you only use up to 2 letters however this is not required. The $ is appended to the front of your variable automatically.

For instance, you might use "MF" as the variable name for "MYGAME.FNT". The "MF" would stand for Mygame Font. The final <@> is optional. Leave it off and you will be guided with prompts telling you to insert your source disk and to insert your destination disk. Leave it on and it will assume the source disk and the destination disk are the same disk.

Finally, step 3 will be repeated until you respond to the request for a SOURCE filename with nothing but RETURN. This means, for example, you may build a LIST file containing several assembly routines to be ENTERed into your mainline programming code at your convenience. To end, simply press Q for Quit.

LINKBAS.UTL -- Cassette Version

To use the LINKBAS.UTL cassette version, position the tape to the beginning of the program and type CLOAD followed by pressing RETURN. When the READY prompt appears, type RUN and press RETURN twice. No filenames are required. Simply position the source tape to read and position the destination tape to write. The only prompts you'll need to answer are for the beginning line number and the varible name you wisn to call your string. One limitation of the

cassette version is the program will only work for one file at a time. To convert another file, the process must be repeated. If you wish to enter a number of these strings into another progam, be sure to have your beginning line numbers staggered accordingly.

MOVMEM

We provide you with a simple assembly language routine to help you move memory around. It can be found in the beginning of the character editor program with the other assembly language routines. You are free to "BREAK" this program and list that portion of the code into a separate LIST file for your own use.

CAUTION: Re-boot the system before going on as the CHARACTER EDITOR performs a great deal of memory management which must be reset for continued use of BASIC. Another approach would be to type in CONT, after listing the assembly routine to a LIST file, then terminate the CHARACTER EDITOR normally. Refer to the BASIC manual on LISTING and ENTERING files.

The routine is called MOVMEM and is held in the STRING VARIABLE MM$. The format of the assembly routine is:

X=USR(ADR(MM$),FROM,TO,BYTES)

where:

ADR(MM$) is the address of the assembly routine.

FROM is the address from which memory is to be moved.

TO is the address to which the memory is to be moved.

BYTES is the number of BYTES to be moved.

Your code might look like this:

```
2000 REM CB=Character Base
2010 CA=CB*256
2020 X=USR(ADR(MM$),ADR(CS$),CA,1024)
2030 REM __Point to new CHARSET
2040 POKE 756,CB
```

Line 2020 MOVED 1024 bytes of memory from the STRING VARIABLE CS$ found at address ADR(CS$) to the address CA, representing the address in RAM where a re-defined character set would reside.

LINKBAS

LINKBAS is one of the most useful utilities for the ATARI computer systems. It was originally designed to take the binary data found in compiled assembly language routines and move it to string variables to be used as USR calls in BASIC programs. It has also proven valuable in performing the same conversion on data files containing "special" information, such as high resolution graphics screens, incarnations to be used in animated PLAYER/MISSILE graphics, and modified character sets. When properly used, LINKBAS can greatly enhance your BASIC programs. It can greatly eliminate clumsy READ/DATA initializations, conserve memory, and improve program speed. Let's start by discussing some simple mechanics of using LINKBAS.

ESI's Simple Instructions:

To get LINKBAS "up & running" you must boot up DOS. Once in DOS simply do a BINARY LOAD by calling function "L". Enter "LINKBAS.OBJ" for the filename. Once LINKBAS loads it will automatically "RUN".

The first prompt you will receive will be for a **destination filename.** The default filename is "D:TEMP.LST" and will appear in parenthesis following the request for a destination filename. If you simply respond with a RETURN then the default is selected. Any existing filename of TEMP.LST on drive #1 will be erased and a new TEMP.LST will be opened. You may respond with another filename if you choose. The filename extension ".LST" is desirable because LINKBAS generates a LIST file. Entering the DRIVE specification is not necessary unless you desire a drive other than number one. If you want a different filename than the default, but, you want the filename extension (.LST) to remain the same, then simply type in your new name followed by a period (.). For example:

D:THISNAME.LST

would be entered as....

THISNAME.

The "D:" would be added to the front of "THISNAME" and because you ended it with a period (.) the extension ".LST" would be appended to "D:THISNAME" to form "D:THISNAME.LST". Once you have entered a destination filename, it will appear at the top of the screen and you will be prompted for a "starting" line number.

The **"starting" line number** is the line number where the LISTed BASIC code generated by LINKBAS will begin. Each successive line number will be incremented by ten (10). The default value of (2000) is in parenthesis following the prompt and, as in the previous prompt, responding with only a RETURN will select the default.

REMEMBER.... LISTED CODE IS "ENTERED" INTO A PROGRAM AND REPLACES ANY DUPLICATED LINE NUMBERS. THEREFORE, YOU WANT YOUR DESTINATION FILE TO CONTAIN LINE NUMBERS THAT ARE NOT DUPLICATED IN YOUR PROGRAM UNLESS YOU INTEND TO "RENUMBER" THE CODE.

Your choice will appear in a small box across from your destination's filename at the top of the screen. As lines of code are generated into the destination file this value will update to the next available line number. It cannot be changed once you have selected a starting line number.

Finally, you are prompted with:

SOURCE/VARIABLE<@>?

As stated previously, this utility was originally designed to convert OBJECT code to string variables. SOURCE filenames follow the same rules as DESTINATION filenames. Disk drive need not be entered unless it is different than drive one. The filename extension need not be declared and a default will be selected if you follow your SOURCE filename with a period (.). Instead of a default filename extension of (.LST) the default becomes (.OBJ) for assembly code OBJECT files. To convert the binary data in the compiled assembly language routine used to move (or copy) the contents of memory from one location to another, we would declare a SOURCE file of:

D:FILMEM.OBJ

would be entered as:

FILMEM.

NOTE - ASSEMBLY LANGUAGE OBJECT FILES TO BE CONVERTED MUST USE THE FILENAME EXTENSION .OBJ BECAUSE THE FIRST SIX BYTES OF AN OBJECT CODE FILE CONTAIN ADDRESS INFORMATION THAT IS NOT PERTINENT TO THE ROUTINE WHEN USED IN STRING VARIABLE FORM. THUS, SOURCE FILENAMES WITH .OBJ EXTENSIONS IGNORE PROCESSING THE FIRST SIX BYTES.

However, there are more requirements to the prompt. Your SOURCE filename should be immediately followed by a slash "/" and a variable name for the string that LINKBAS will generate containing the data in the SOURCE file. I recommend variable names no longer than two characters to conserve memory. For example:

D:FILMEM.OBJ/FM

would be entered as:

NOTICE that I did NOT add a "$" to the variable name. LINKBAS will do that for you.

The prompt is almost satisfied. At this point you have two options. If you simply hit RETURN you will be prompted to insert the SOURCE DISK, the file will be read and converted, then you'll be prompted to insert the DESTINATION disk, and the code will be LISTed to the destination file. This procedure assumes your source file may be on a disk other than the destination file. I rarely have that situation. I try to keep all my short assembly language routines on one disk and create a TEMP.LST source file on the same disk. Once the file is completed I simply transfer it to my workdisk. The second option is more suited to this kind of operation. Simply add a "@" to your response before hitting RETURN and LINKBAS will assume single disk operation. The LESS THAN and GREATER THAN symbols that bracket the "@" in the SOURCE/VARIABLE<@> prompt serve only to indicate the "@" is an optional request. For example:

FILMEM./FM@

will automatically OPEN the source file FILMEM.OBJ on disk drive one, read the contents, convert it, CLOSE the source file, and LIST the converted data to the destination file. All this is done without having to satisfy SOURCE DISK/DESTINATION DISK insertion prompts.

Once you have satisfied a SOURCE/VARIABLE<@> prompt the entire results will be posted on the screen and conversion will proceed. Using our same example:

FILMEM./FM@

would get posted to the screen as:

D:FILMEM.OBJ/FM$

Termination of LINKBAS is accomplished by simply responding to the SOURCE/VARIABLE<@> prompt with a RETURN. You will be returned to DOS.

APPLICATIONS OF LINKBAS

ASSEMBLY ROUTINES IN BASIC PROGRAMS

We have already touched on the assembly language routine. I will go into a little more detail. Let us suppose I want to create an assembly language routine that will always generate consistent time delays in my BASIC programs, regardless of where the delay is situated in my code. The first step is to write such an assembly language routine using one of several EDITORS available for the ATARI. A 6502 ASSEMBLER will allow the code to be compiled to an OBJECT (.OBJ) file on disk. When I compiled the source file I named it TDELAY.OBJ for "TIME DELAY". Now I need to get it into my BASIC program, so, I run LINKBAS.

When I write a program, I start all my "mainline" code at line 4000. Lines 2000-3999 are reserved for assembly language routines and other string initializations whether they are used or not. This way, I know that string variables created in a LIST file by LINKBAS using the default starting line number of 2000 will ENTER into my program without conflicting line numbers. In case you're wondering what happened to lines 1000-1999, they are reserved for the program HEADER and VARIABLE DICTIONARY. Line numbers below 1000 are not used because they indent and detract from uniform readability. High legibility is a landmark of good professional coding. That is why I also avoid multiple line statements and lines that exceed 37 characters causing them to wrap around. LINKBAS was designed to do the same thing. When a line of code is LISTed to the destination file it never exceeds 37 characters. For example:

```
2000 DIM TD$(55):REM __FILMEM.OBJ
2010 TD$(1)="XXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

would look like:

```
2000 DIM TD$(55):REM __FILMEM.OBJ
2010 TD$(1)="XXXXXXXXXXXXXXXXXXXXXXXXX"
2020 TD$(24)="XXXXXXXXXXXXXXXXXXXXXXXX"
2030 TD$(46)="XXXXXXXXXX"
```

This makes it much easier to locate and change any character in the string. It is also much neater and the cost in extra memory for each additional line number is negligible compared to the benifits.

APPLICATIONS OF LINKBAS

When I run LINKBAS I respond to the destination filename prompt with a RETURN thereby selecting the default filename TEMP.LST. I also respond to the starting line number prompt with a RETURN selecting the default line number 2000. When prompted for the SOURCE/VARIABLE<@> I enter:

```
TDELAY./TD@
```

and the source file TDELAY.OBJ is opened, read, converted, and LISTed to TEMP.LST. The next time I receive the SOURCE/VARIABLE<@> prompt I simply respond with a RETURN to terminate the program.

Getting the LISTed code into my program is easy. I just get into BASIC, LOAD my mainline code and

ENTER"D:TEMP.LST" into my program. Now I can call the routine anywhere in the BASIC program like this:

```
4220 U=USR(ADR(TD$),300)
```

where: ADR(TD$) is the starting address of the assembly routine that was initialized at line 2000, and the 300 is a value passed to the routine when I performed the USR call. In this case the 300 represents 300 jiffies (1/60ths of a second) or a 5 second time delay.

SAVING SCREEN GRAPHICS

Let's look at another example of how LINKBAS can serve us. Suppose we want to show a fancy graphics screen as a title page or playfield. The procedure to convert such a sceen to a string variable is much the same as converting assembly routines. Creating the source file is entirely different. See the discussion on CIOCTL for this exact procedure. Meanwhile, assume a source file is in existence. This file is truly a data file and differs from an assembly object file in that the first six bytes of the file are useful data. THEREFORE - SUCH SOURCE FILES SHOULD NEVER HAVE THE FILENAME EXTENSION .OBJ. For a graphics screen file I usually use the filename extension .SCN. Now when prompted with SOURCE/VARIABLE<@> I might respond with:

```
PLAYFLD.SCN/PF@
```

This would cause the entire screen contents to be saved in a string variable. This can be fast and efficient for screen graphics but can be expensive in terms of memory if it is a hi-res graphics mode and you are trying to fit your program into less than 16K.

Once the screen is incorporated into your BASIC program it can be displayed in a flash with MOVMEM.OBJ which is included in this tutorial. An example might look like this:

```
4060 GRAPHICS 0
4070 SCA=PEEK(88)+PEEK(89)*256
4080 U=USR(ADR(MM$),ADR(SC$),SCA,960)
```

Line 4080 simply moves 960 bytes of data from the string variable SC$, which contained the desired screen, to memory beginning at SCA which is the beginning address of the screen for the current graphics mode. The screen changes instantly, and avoids the time it would take to "draw" the desired display. Because of such speed the same technique can be used for PAGE FLIPPING. To FLIP back and forth between two different screens might be done like this:

```
4060 GRAPHICS 0
4070 SCA=PEEK(88)+PEEK(89)*256
4080 U=USR(ADR(MM$),ADR(S1$),SCA,960)
4090 U=USR(ADR(TD$),600)
4100 U=USR(ADR(MM$),ADR(S2$),SCA,960)
4110 U=USR(ADR(TD$),600)
4120 GOTO 4080
```

This simply FLIPS between screen #1 held in string variable S1$ and screen #2 held in string variable S2$. Lines 4090 and 4110 are time delays so the screens can be viewed for 10 seconds each. This program will run until interrupted by pushing the BREAK key.

PLAYER/MISSILE ANIMATION

LINKBAS can be used to store the various incarnations needed to generate animated characters in PLAYER/MISSILE graphics. Let us suppose it takes three different drawings to make a stick figure look like it is walking. The sequence of motion for the character might be:

1. Left foot out.
2. Both feet standing.
3. Right foot out.
4. Both feet standing.

Then back to #1.

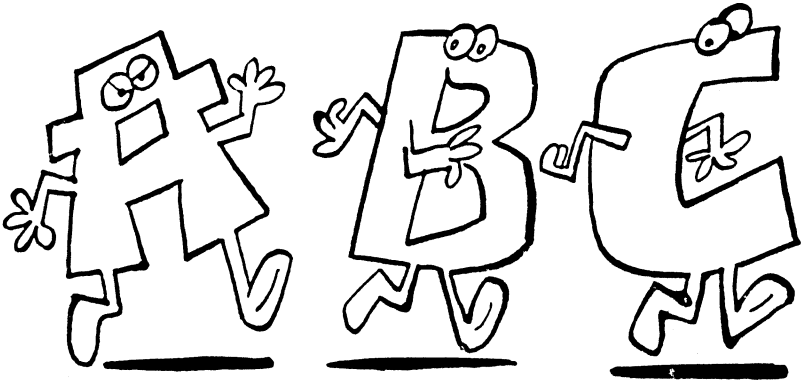
Since #2 and #4 are identical we have only three incarnations of the walking figure. To create a source file of these figures we could use the character editor included in this Tutorial, and draw them as follows:

1. Replace the (!) with a figure having both feet standing.
2. Replace the (") with a figure having the left foot out.
3. Replace the (#) with a figure having the right foot out.

When the re-defined character set is saved (D:WALKING.FNT for this example) the first four characters will be:

1. SPACE
2. BOTH FEET STANDING CHARACTER
3. LEFT FOOT OUT CHARACTER
4. RIGHT FOOT OUT CHARACTER

FOLLOWED BY THE REST OF THE CHARACTER SET.



LINKBAS

We only need characters 2 through 4 for our incarnations. This is how we would get them into our source file for conversion by LINKBAS:

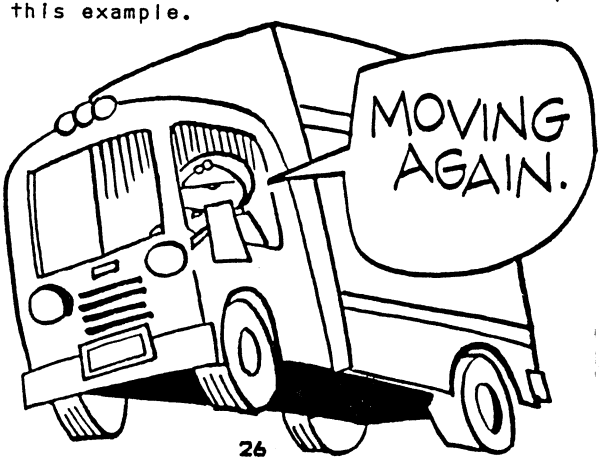
without line numbers - type in the following in BASIC:

```
OPEN #1,4,0,"D:WALKING.FNT"
 (opens the character font file)

FOR I=1 TO 8:GET #1,B:NEXT I
 (reads away the first character which is a SPACE)

OPEN #2,8,0,"D:WALKING.SCE"
 (creates and opens our source file)

FOR I=1 TO 24:GET #1,B:PUT #2,B:NEXT I
 (moves the next three characters from the character
font file      to the source file.  Three characters X 8
bytes each      equals 24 bytes)

END
 (closes all files)
```

We now have a source file for the PLAYER/MISSILE incarnations. What is next? Well, LINKBAS, of course! When prompted SOURCE/VARIABLE<@> we just enter:

WALKING.SCE/PM@

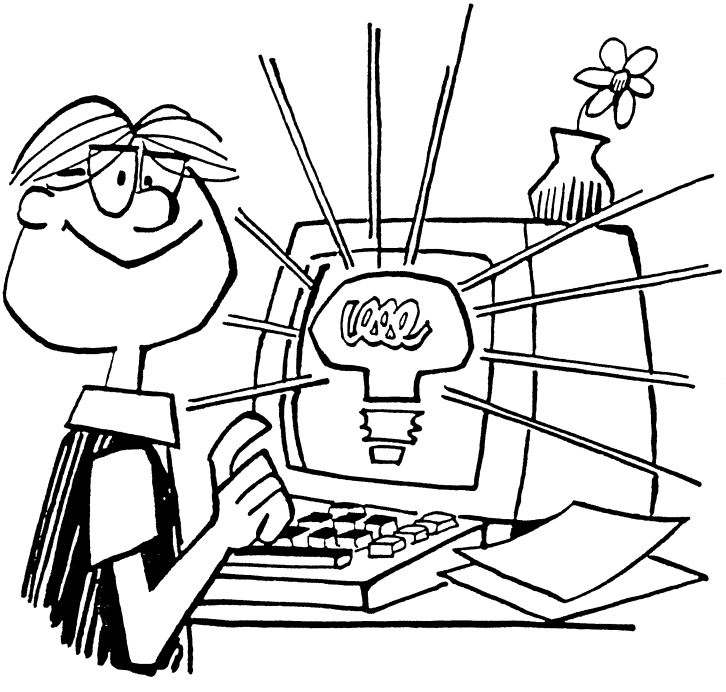
This will move the 24 bytes found in the source file D:WALKING.SCE into the string variable PM$ (for Player Missiles) in the destination file without prompting for source and destination disks (@). We can then ENTER the destination file into our mainline program to use with PLAYER/MISSILE graphics. See MOVPMS for a description of implementing this example.



ALTERNATE CHARACTER SETS

This is easy! Simply create a source file using CHAREDIT.UTL supplied in this tutorial and use LINKBAS to create your LIST file. ENTER your .LST file into your BASIC program and away you go! See MOVMEM for an example of how to install alternate character sets using this technique.

By now you should be full of other uses of LINKBAS for your own programming needs. Good! I just ran out of ideas, so I'll stop here. We love to look at your brilliance, so feel free to send in your ideas. You may even end up in a future edition of this lesson!



FONT PRINTER

It's easy and convenient to edit fonts on the screen using CHAREDIT, but what if you want a more permanent record of the way each character is designed? With FONTPRINT, you can have a hardcopy printout of as much of your font as you want. You can then use this printout as a permanent record in case you lose your font, or just as a scratch sheet to make changes to the characters before using CHAREDIT.

HOW TO LOAD FONTPRINT

WARNING!

Fontprint uses the Basic cartridge to run. If you want to print out the shapes of a character set in memory, DO NOT turn off the computer to load fontprint. You don't want to lose the data in memory, so be sure to have the Basic cartridge in ahead of time, and then load as follows:

CASSETTE:

Make sure your Basic cartridge is in place. Put the FANCY FONTS cassette in the cassette player, position the tape to the beginning of FONTPRINT, press PLAY, type 'LOAD "C:"', and press RETURN. After the Atari beeps, press RETURN again. After the program has loaded and the READY prompt appears on the screen, type 'RUN' and press RETURN.

DISKETTE

Make sure that the BASIC cartridge is inserted. Turn on your printer (and interface if you need it). If you're using disk, turn on the disk drive, insert the FANCY FONTS diskette, type 'RUN "D:FONTPRNT"' and press RETURN.

HOW TO USE FONTPRINT

The first thing that FONTPRINT needs to know is:

Load font from what file?

If the font you want to print is already in the Atari's memory, then just press RETURN. You can tell if the font is in memory by looking at the screen. The font you see is the font you get. If you want to load a font from disk type in :

D:XXXX

where XXXX is the name you used to save your character set on the disk. Cassette users input:

"C:"

FONT PRINTER

The screen will then show:

LOADING FONT FROM: <your filename>

while it loads in your font. If the Atari can't find your file it will print an error message. If this happens you must type in the filename again. Make sure that you spell it correctly, and that you're using the right disk or tape! Your tape must be at the exact spot that you started to save the data at.

Assuming that everything loads ok, you'll next be asked:

Starting and ending characters to print (0,127 to print all)?

If you want to print the whole font, type '0,127' and press RETURN. If you only want part of the font, type the numbers of the first and last characters you want printed, separated by a comma, and press RETURN ('12,12' will print character 12 only). You should note that the character numbers you type here are **not** the same as the ATASCII (CHR$) numbers. The Atari stores characters in a different order than ATASCII. See your Basic manual for reference charts and explanations. The following chart will help you convert from one to the other (you can find a chart of characters and their ATASCII values in your BASIC manual):

For ATASCII values:	Make this change:
0-31	add 64
32-95	subtract 32
96-127	no change

So, for example, ATASCII values 0 to 40 would correspond to Atari values 64 to 8, thus you would type '8,64' followed by RETURN.

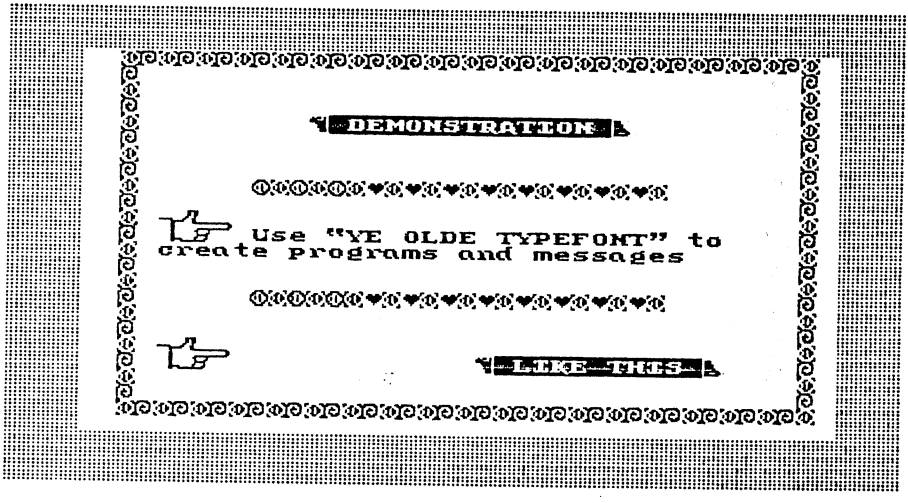
After you have decided which characters you want printed, you'll be asked;

Do you want to print on a 40, 80, or 132 column printer?

Type either '40', '80', or '132' and press RETURN. The Atari will now print your font for you, using a '*' to indicate dots that are turned on in a character, and a blank space to indicate ones that are turned off. If you would like to have it print something other than '*' and ' ', you can change B$ and C$ in line 20 of the program.

FONT DEMO

Ok, now we can design fonts and print them and all that, but what do we do with them? The obvious use of a custom font is to make textual displays look better. Too many people, however, limit their use of fonts to just text. Don't forget the graphics characters. They can also be used to brighten up textual diplays. You can use them to make fancy borders, or highlight specific phrases in creative ways (the word of the day is "creative"). FONTDEMO is a simple example using the TYPEFONT font. RUN it from BASIC and look at the display, then press any key to see what characters have been redefined to make up the display. Press BREAK and, holding down the CTRL key, type "HAND". Now, use CTRL and the arrow keys to get underneath the first character you just typed, and (still holding down the CTRL key) type "PLOT". That's how to combine characters to get a larger character! Play around a little with the different CTRL characters and then rerun TYPEFONT.DMO and some of our craziness will not look so difficult to program as it first appeared.



TYPEFONT PROGRAM

TYPEFONT.PGM

The Running Program contains the typefont, special characters, and program logic to support the GOSUB commands. In other words, the Running Program gives you the tools to create and save your own artistic presentations.

A. Load the Running Program as described earlier. For disk users it is called TYPEFONT.PGM

B. Type 'RUN' and press RETURN to initialize the Running Program.

C. You will see the screen go black and after a few seconds a hand will appear on the screen, accompanied by a bird chirp and the word "READY". This only happens the first time you run the program after loading it. It is your signal that the special characters are all loaded in memory and the Atari is READY for you to add the instructions for your own program.

D. Type RUN and press RETURN again. Just the word "READY" will appear. After you load and run the Running Program the first time, the Atari is ready to run **your** instructions when the RUN command is used again. After completing your instructions, it tells you it is READY to RUN again. Since you haven't given it any instructions yet, it can only tell you that it's READY.

Helpful Background

A. **LINE NUMBERS** Use any line number from 10 to 29984. DO NOT USE line numbers 0-3 or any line number larger than 29984 because the Running Program uses these lines. Don't use an END statement either.

B. TRY THIS:

```
100 GOSUB CHAND
110 U$="BEHOLD!":GOSUB SCROLL
```

Run the program. Congratulations, you can now write your own "Eternal Truths"!

THIS IS SOME FUN

C. **LISTING YOUR PROGRAM** Run the above program again. When the "READY" prompt appears, press the SELECT button. Press SELECT again

1. Now type "LIST" or "L." and press RETURN (a little too much?)
2. Sorry, SELECT won't work now. RUN your program and then press SELECT again.
3. Try L. 10,2000
Try L. 100
4. Try L. TIME (If you want particulars on how any of the GOSUB routines work, just "LIST" them. Most of them take several lines.)

Try L. HAND,HAND+10 (All subroutines end at the first RETURN statement encountered (except GOSUB TEXT.)

D. **GETTING OUT OF TROUBLE**

1. Press SYSTEM RESET. Run your program, or...
2. Press SYSTEM RESET. Now type "GOSUB OLDTYPE" and press RETURN. You are now back in control.
3. If you get into a situation where none of the above work, you have overwritten one of the Running Program lines and will have to reload the program.

E. **KEYSTROKES TO PRODUCE SPECIAL CHARACTERS**

1. The direct command GOSUB KEYCODE will display the special characters in YE OLDE TYPEFONT and the keystrokes required to print them to the screen.

F. **ERRORS — "SOMETHING IS AMISS!"**

1. Deliberately program an error and run the program. For instance, "200 GGG" followed by RETURN will show an error on the screen. Run the program anyway. Fun?
2. You can use the screen editor to correct the listed line if you wish (of course, the listed line is not always the real cause of the error).
3. The variable values listed are for variable names that you might use in writing your own program. The automatic listing of their values can help you interpret error messages.

G. **You are now ready to ADD YOUR OWN PROGRAM** (as discussed in the next section). The idea is simple. The Running Program is a series of instructions on numbered lines 0-3 and 29994 to 32310. You are free to write your own instructions on any line number between 4 and 29993. If you are a beginner, you can forget about the details of the support program, as it diverts the computer to your instructions.

WRITING A SAMPLE PROGRAM

A. **RUN, then press SELECT.** Then delete all listed lines by typing the line number and pressing RETURN.

B. **RUN THIS:**

```
120 U$="CAUTION":GOSUB MSCROLL
```

("M" means "middle"). The word "CAUTION" won't blink when the START button is pressed.

C. **BLINKING** Retype the word "CAUTION" in line 120 so it is in inverted characters. To get inverted characters, press the reverse key (see your Atari manual) before you type the word, and then again afterwards. Run the program again. The START button will now make the inverted characters blink.

D. **TEXT** Add these two lines to your program and RUN:

```
140 U$="An ancient philosopher once told us that
 people in glass houses shouldn't throw ......"
150 GOSUB TEXT
```

E. **THE PUNCH LINE** Add this line and RUN:

```
180 U$="HAND GRENADES":GOSUB BALL
```

F. **SOUND** Add this line and RUN:

```
190 GOSUB GONG
```

G. **TIMING** For a little suspense, add this line and RUN:

```
160 GOSUB TIME
```

H. **AUTOMATIC BLINKING** Change line 160 to this and RUN:

```
160 GOSUB BTIME
```

TYPEFONT PROGRAM

I. **VERTICAL SPACING** Use the BASIC "PRINT" command (abbreviated as "?") to add blank lines for vertical spacing. Add these lines and RUN:

```
110 ?:?
130 ?:?
170 ?:?
```

J. **BORDERS** If you want a border, specify it at the beginning of your program. Add this line and RUN:

```
100 GOSUB BORDER
```

K. **FINISHING OFF** If your program uses a border, you must tell the computer when the picture inside the border is finisned. Add this line and RUN:

```
200 GOSUB EXIT
```

Saving and Reloading Programs

A. **SAVING A PROGRAM**

1. Put a cassette or disk into your recorder or drive (position the cassette to a blank spot).
2. If you're using cassette, press both the PLAY and RECORD keys.
3. Press SYSTEM RESET on the computer. If you're using cassette, type "LP." and press RETURN (an "ERROR 138" message will appear on the screen. Ignore it. Taking this extra step makes CSAVE more reliable).
4. Type "CSAVE" or 'SAVE"D:filename"' (depending on whether you're using cassette or disk respectively. If you're using disk, replace "filename" with the name you want to save your program under) and press RETURN. If you're using cassette, the Atari will beep twice. Press RETURN again after it does.
5. A new "READY" prompt will appear when your program has been saved.

B. **REUSING A PROGRAM YOU HAVE SAVED**

Just CLOAD or LOAD"D:filename" from the cassette or disk you saved the program on. You can then RUN it and change or modify it without having to refer back to the master program cassette/disk.

TYPEFONT PROGRAM

Large Letters

A.GRAPHICS MODES

This program is best suited to text modes GRAPHICS 0, GRAPHICS 1, and GRAPHICS 2 provided by the Atari. Unless you direct the Atari otherwise, it will automatically go to GRAPHICS 0. It is sometimes fun to use the larger letters of the other graphics modes to add more emphasis to your messages.

B. SAMPLE PROGRAM

Try this program:

```
10 GRAPHICS 2:GOSUB OLDTYPE
20 POSITION 5,7
30 ?#6; "HELLO"
200 STOP
210 GRAPHICS 2:GOSUB OLDTYPE
```

When the Atari stops at line 200, type "CONT" and press RETURN.

C. EXPLANATION

Line 10 tells the Atari to go to graphics mode 2. When you change graphics modes, the Atari will automatically go back to the regular typefont, so if you want to use the redesigned letters in YE OLDE TYPEFONT, you must call them up with GOSUB OLDTYPE.

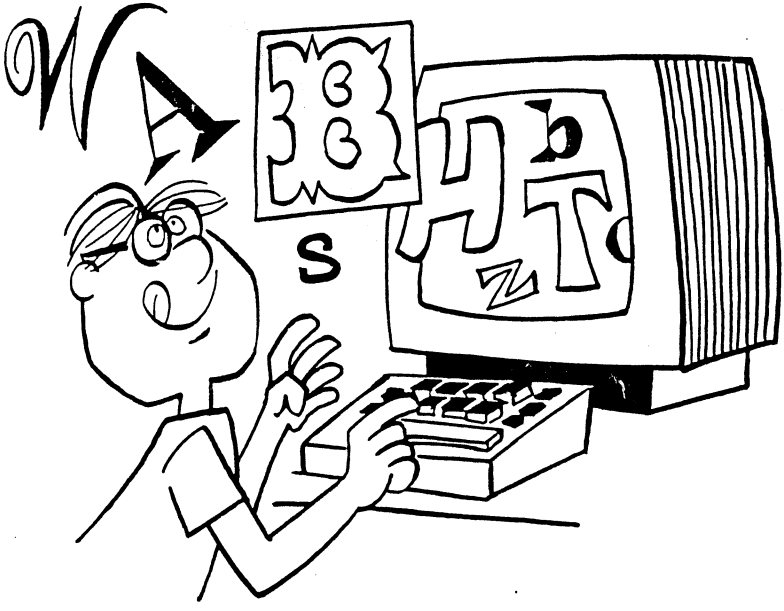
Line 20 centers the word we want to print on the screen. In graphics mode 2, the screen has room for twelve lines of text (numbered 0-11) and twenty large letters on each line (numbered 0-19). The word "HELLO" can be centered on the screen by starting with the "H" at line 5, column 7. This is done with the position command. In this case, we use POSITION 7,5.

Line 30 prints the word "HELLO" on the screen. The BASIC command to print letters on the screen in the large text modes is "?#6;" instead of just "?" (don't forget that "?" is an abbreviation for "PRINT").

Line 200 stops the program so you can examine it to see what changes you want to make.

TYPEFONT PROGRAM

Line 210 returns the Atari to graphics mode 0 so you can modify your program if you want to. You can make attractive presentations in graphics mode 2, but it is difficult to write program instructions while you are in this mode. Therefore, you should be sure to include instructions in your program that will make it easy to return to graphics mode 0. When you are satisfied that you have what you want on the screen, then change line 200 to "200 GOSUB TIME".



D. LIMITATIONS

Graphics modes 1 and 2 will only print capital letters and punctuation marks on the screen (although you can try POKE 756,226 to get lowercase and graphics only). However, they do give more control over text color. For example, change line 30 in the above program example so that "HELLO" contains uppercase, lowercase, and inverse video. Notice how each one is a different color when you RUN the program.

E. BLINKING TEXT

These modes also don't support inverse characters (you saw that above), so you can't use the blink feature. You can, however, achieve a blinking effect by switching back and forth from Atari letters to YE OLDE TYPEFONT letters. Add this line to the above program to illustrate this effect:

```
40 GOSUB SWTYPE
```

F. COMBINING LARGE AND SMALL LETTERS

Graphics modes 1 and 2 are split screen modes. This means that in addition to the large text in the upper part of the screen, there is also four lines of graphics 0 text at the bottom of the screen, in a little "window". All of YE OLDE TYPEFONT special characters will display in this window, although it is to small to use the GOSUB BORDER command. To illustrate the effect of the window, add the following line to the above program:

```
50 U$="EVERYBODY":GOSUB BALL:GOSUB GONG
```

Note: Once you have changed to graphics mode 1 or 2, all large letters appearing on the upper part of the screen must be printed with the "?#6;" command as described above.

Special effects such as hands, scrolls, etc will appear in the small four line graphics 0 window at the bottom of the screen. While in modes 1 or 2, you may also write normal size text to this window by using the regular "?" (PRINT) command.

Color

The SETCOLOR Command

You can change the colors in your display by using the SETCOLOR command at the point where you want the change to happen. The command looks something like this:

SETCOLOR R,C,B

A. The number specified by R (0-4) chooses the color register you want to change. Each of the five color registers controls the color of a different part of the screen display.

B. The number specified by C (0-15) chooses the color that you want to change to. For example, 0 means black, 2 orange, 3 red, 8 blue, and 12 green (see your BASIC REFERENCE MANUAL for a complete list of the colors associated with each number)

C. The number specified by B (0-14, even numbers) chooses the brightness of the color you want to change to. For example, 0 would give a very dark color, 14 would give a very bright color.

Changing color in graphics mode 0

The colored characters in YE OLDE TYPEFONT are produced by a technique called "artifacting" (use CHAREDIT to look at OLDE.FNT to see how this is done). Colors produced by artifacting show up best against a black background. You can, however, present these "artifact" colors in pastel shades against various other dark backgrounds (always specify a brightness value of zero to prevent washed out colors in this case).

TYPEFONT PROGRAM

A. BACKGROUND COLOR
The background color in graphics mode 0 is controlled by color register 2. Try this:

```
100 GOSUB BORDER
200 SETCOLOR 2,8,0
300 GOSUB EXIT
```

Change the value of the second number (8) in the SETCOLOR command to get different background colors (remember to use a value between 0 and 15).

B. OUTER BORDERS
You can produce a solid color outer border around your text by changing color register 4. Try this:

```
100 GOSUB BORDER
200 SETCOLOR 4,3,8
300 GOSUB EXIT
```

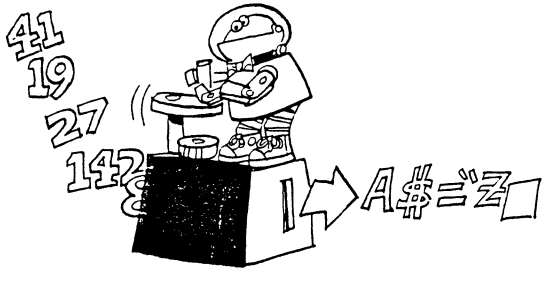
Change the value of the second number (3) in the SETCOLOR command to get different outer border colors.

Changing color in graphics modes 1 and 2 (large letter modes)

The five color registers control different parts of the picture in graphics modes 1 and 2 than they do in graphics mode 0.

COLOR REGISTER	PART CONTROLLED
0	capital letters
1	lowercase letters
2	inverse capital and graphics 0 window
3	inverse lowercase
4	background and outer border of main screen

Specifying lowercase and inverse characters controls the color of the characters but they will still appear in normal capital letters.



TYPEFONT PROGRAM

A. EXAMPLE

```
100 GR.2:GOSUB OLDTYPE
Print line 150 with Lister
```

Run this, then type "SETCOLOR 0,3,0" and press RETURN. Try changing the value of the first number in the SETCOLOR command (from 0 to 4) to see the different parts of the picture that the different color registers control.

B. PROGRAMMING TIPS

1. Make sure you have high contrast between the letters and the background. In other words, if you use a low brightness for the backgorund, then use a high brightness for the letters and vice-versa.
2. Use low brightness for the graphics 0 window (controlled by register 2) to assure that "artifact" colors show clearly. Incidentally, the color registers are **all** automatically reset to their default values when you switch graphics modes.

Borders

The GOSUB BORDER command creates a border with the characters in the string UB$. The horizontal borders are created by repeating all four characters in the string. The vertical borders are a repeat of the first two characters in the string. Any characters that will print to the screen (including the ESC key) can be used in UB$ to create a border.

If you do not specify UB$, the Atari will randomly select a border for you. If you wish to control the border pattern, then make sure you specify the characters you want in UB$ before you use GOSUB BORDER (type "GOSUB KEYCODE" and press RETURN to see a list of the characters available). Try the following using various characters in UB$:

```
10 UB$="1234"
20 GOSUB BORDER:GOSUB EXIT
```

Repeat the experiment with different characters in UB$.

Inverse characters in UB$ will blink when the BON or BTIME subroutines are called. The combinations are almost infinite, so let your imagination run wild!

TYPEFONT PROGRAM

Timing

As you program your presentation step by step, you'll want to create pauses so that those watching can have time to read messages, or maybe you just want a pause for dramatic effect. For your convenience, the following commands provide a five second delay: GOSUB TIME, GOSUB BTIME, GOSUB SWTYPE. The commands GOSUB BIRD and GOSUB GONG provide a shorter delay.

The duration of the GOSUB TIME command can be altered by changing the number 20 in line 20056 to a larger or smaller value (20 is roughly five seconds, so 4 would represent close to a second). For those moments when you need a special delay, use a FOR/NEXT loop:

```
100 FOR DELAY=1 TO A:NEXT DELAY
(adjust the value of A to get the delay you want)
```

The blink rate can be altered by POKEing a value between 0 and 4 into memory location 1724:

```
POKE 1724,0        normal blink rate
POKE 1724,1/2/3    faster blink rates
POKE 1724,4        no blink
```

If you POKE any value greater than 4, the program will not work properly.

Long (multiple frame) presentations

Simply program one presentation after another. Conclude each segment with a delay and then make sure each starts with a command that clears the screen (GOSUB BORDER, GOSUB CMSCROLL, GOSUB CMTEXT, and GOSUB CHAND all clear the screen). Don't forget that a GRAPHICS command also clears the screen.

YE OLDE TYPEFONT will automatically break long text presentations into frames for you when text is written inside borders. For example, you can present the Gettysburg Address in a series of frames as follows:

First, GOSUB BORDER (use alternating CTRL-Z and CTRL-Q characters in UB$ for a great "Lincolnesque" border).

Next we need a title: U$="Gettysburg Address":GOSUB MSCROLL:?

Finally, put as much text as you can into U$ (about three lines) and GOSUB TEXT on the next program line. Repeat this process until the address (or whatever) is complete, then GOSUB EXIT.

When the program is run, it will automatically divide the message into frames. It will then display each frame for five seconds and then draw a new border for the remaining text. It will repeat this process until a GOSUB EXIT is encountered.

For paragraph separation, finish U$ at the end of the paragraph and add a "?" command after GOSUB TEXT.

For paragraph indentation (if desired), place an appropriate number of spaces at the beginning of the next U$ specification.

Once you have created your presentation, you can make it run forever by making the last command "POKE 77,0:RUN".

If your presentation is only a few frames long, let the Atari specify the borders. That way, even though the message will be repetitive, the constantly changing borders will help hold the viewer's attention.

The length of the presentation you can make depends on your Atari's memory size. It works out to about 15-20 frames for every 8K of memory other than the first 8K (that's used for the program).

Sound (music and recorded commentary)

You can add symphonic music, rock and roll, or even your personal commentary to your presentation, and it's easy!

The 410 recorder will play music cassettes (or any type of prerecorded cassette) over your TV speaker.

The command POKE 54018,52 turns on the recorder, and POKE 54018,60 turns it off.

Prepare your program in the normal manner. When it is working to your liking, add the above POKE commands at the points you want the music or commentary to start and stop.

Save your program to cassette or disk in the normal manner. Reload the program. Before you run it, place the separate music or commentary cassette in the 410 recorder and press the PLAY button. It's as easy as that!

As long as you use a normal bias cassette (it'll usually say on the cassette whether it's normal bias or not), it doesn't matter whether the tape was recorded in mono or stereo. The Atari will just ignore sound on one track.

To prepare a commentary cassette, use a portable cassette recorder (or a stereo cassette deck) with built in or external microphone. Set the recorder close by and RUN your program. Start recording at the point in the program you want the commentary to start, and simply make the commentary while watching the program. You can even play a little background music on your stereo if you want to.

Precise synchronization (matching the sound to the picture) is impossible, but general commentary will almost always provide impressive results.

You can also record sound on the end of your program cassette, right after the program. To do this, save your program to cassette, then as soon as it's done saving, put the cassette in the portable recorder for your commentary (be careful not to change the position of the tape). This way, you can just load the program from cassette and RUN without having to switch cassettes.

Creating moods and a historical timeframe

The characters in YE OLDE TYPEFONT are carefully selected to be versatile.

A. MOODS
Moods are easy. Use hearts and flowers for cheery, friendly moods. Use bright colors and bird chirps with friendly, positive text. For somber responses, go to drab colors and gongs. For humor and flamboyance, it's easy to exaggerate and do the unexpected with hands, birds, gongs, and timing. For serious presentations, be tasteful. Use simple borders, create an accurate time frame, and take it easy on the blinking. Music or commentary can also add a professional touch.

B. TIME SETTING TIPS
The effect of poetry, proverbs, historical information, limericks, and messgaes you create yourself can often be dramatized by setting your presentation in an appropriate time period.

TYPEFONT PROGRAM

1. **ROMAN EMPIRE:** Use simple borders (try UB$ made up entirely of inverted spaces). Use all capital letters. Use Latin mottoes. Date your presentation in Roman numerals.
2. **MIDDLE AGES:** This was a period of illuminated manuscripts and heavy church influence. Illuminate often in blue and gold. GOSUB BORDER with the default borders is ideal. Use caps and lowercase. Date your presentation "Anno Domini". Use Latin mottoes. Manuscripts frequently begin with a capital letter imbedded in a block of decorative illumination. This effect is easy to duplicate using CTRL-E. In the following example, "o" represents CTRL-E, "@" reverse CTRL-E, and "-" ESC CtRL-<right arrow>:

```
10 GOSUB BORDER
20 ? "--@o@"
30 ? "--oVo ini, Vidi, Vinci"
40 ? "--@o@"
50 GOSUB EXIT
```

A certain "bawdiness" persisted in the fourteenth and fifteenth century (remember Chaucer?). Use Middle English.

3. **THE 17th CENTURY (1600's):** Date your work with Arabic numerals. Colorful illuminated borders are fine. The influence of the church was strong in the first half of the century and weaker later. For early periods, use A.D. with dates. A little obsolete or poetic English helps (this was Shakespeare's time). Stern messages with Godfearing moralizing were in, but became softer and more romantic later.

4. **THE 18th CENTURY (1700's):** Date your presentation. Use simple borders and color sparingly. This was an era of artistic accomplishment that is hard to duplicate without a whole 'nother character set. Graceful curves in borders using CTRL-Q characters can help if combined with messages proclaiming lofty, positive ideals. Homey, practical proverbs later in the period. Also sparing use of the hand and scroll with inverted text inside.

5. **THE 19th CENTURY (1800's):** Things began seriously enough, but became rather rambunctious as time wore on. Use conservative borders (a repeat of CTRL-Z, CTRL-Q in borders is typical) and pretentious words for early periods. The latter half of the century is the most fun and a great period to set attention-getting messages in. It gives you the excuse to go bananas! Use hands, scrolls, gongs, birds, blinking text, and color with mad abandon to get your point across. This was the age of commercialism, flamboyant advertising, exaggerated claims and over-embellishment. Anything goes!

DRAWING PICTURES

YE OLDE TYPEFONT is basically a text manipulation program. Though it has no special features to support drawing pictures, the special characters have been given letter assignments that make them easy to remember (see the section on the character set). For simply messing around, type "BYE" and press RETURN. Use the cursor controls to position yourself on the screen and try plotting a few characters. This is an easy way to get some ideas for a picture.

If you develop an idea you want to save, you can program a picture by printing one line at a time, starting at the top of the picture and working your way down. The "Title Page" and "Nifty Pictures" frames in the demonstration program were done this way, and you can refer to the demonstration program listing to get the idea. This technique requires some try and miss experimenting and skillful use of the text editor to get things exactly the way you want them, but it's usually worth the effort.

TYPEFONT PROGRAM

NOTE FOR BEGINNING PROGRAMMERS

If you are getting into programming in any depth, the first thing you should do is master the terrific text editor that Atari BASIC provides. Consult your BASIC manual to find out how to use the arrow keys, CTRL-INSERT, CTRL-DELETE, SHIFT-INSERT, SHIFT-DELETE, and the BACKSPACE key to gain greater control over the Atari screen.

Graphics 8

For more adventuresome programmers, the special characters of YE OLDE TYPEFONT can be plotted to the GRAPHICS 8 screen. In order to be able to do this, however, you must have at least 24K of memory (32K with disk).

Here is a listing of the program G8PLOT. It creates a subroutine that will plot characters to a graphics 8 screen. (What more do you need?).

```
4     GPLOT=5:U4=40:DIM D$(U1):GOTO 13
5     X=PEEK(85)+U8xPEEK(86)/8:Y=PEEK(84)
6     UA=PEEK(88)+U8xPEEK(89):UB=UA+U4xY+X:FOR UD=U1 TO
LEN(U$):D$=U$(UD,UD):GOSUB 9:UC=U8xPEEK)1790)+8xX
7     FOR UN=UO TO 7:POKE UB+U4,PEEK(UC+UN):IF UE=U1 THEN
POKE UB+U4xUN,U8-PEEK(UC+UN)
8     NEXT UN:UB=UB+U1:NEXT 85,PEEK(85+8):POKE 84,21:RETURN
:REM Subroutine Exit
9      UE=UO:X=ASC(D$):IF X>127 THEN X=X-128:UE=U1
10     IF X>31 AND X<97 THEN X=X-32:RETURN
11     IF X<32 THEN X=X+64:RETURN
12     RETURN
13     REM: END OF G8PLOT
32118 PEEK(106):UB=U8x(UA-40):POKE206,UA-40:POKE
205,UO:UC=U8-U1:RESTORE  32300:REM(Changes Running Program)
20     REM: SAMPLE APPLICATION STARTSAT LINE 100 & ENDS AT
LINE 150
100    GRAPHICS 8:GOSUB OLDTYPE:REM FIRST call GR.8 & OLDTYPE
110    COLOR 1:PLOT 180,20:DRAWTO 100,100:DRAWTO 240,100:REM
THEN what do you want in GR.8
121    POSITION 20,70:REM: THEN position first CHARACTER you
want overplotted(40 X 160 Grid)
122    U$="45°ANGLE  @ ":REM THEN putmessage and graphics
characters you want plotted in U$
123    GOSUB G8PLOT:REM: THEN GOSUB G8PLOT to plot to screen
124    POSITION 8,130
125    U$="]] GEOMETRY CAN BE ]]"
126    GOSUB G8PLOT
130    U$="FUN":GOSUB BALL:GOSUB GONG:GOSUB BTIME:GOSUB
TIME:REM OTHER   SUBROUTINES appear in TEXTWINDOW at bottom
150    GRAHPICS 0:GOSUB OLDTYPE
```

TYPEFONT PROGRAM

As you can see, G8PLOT takes up lines 4-13, so make sure not to use these if you're using G8PLOT in your program.

G8PLOT examines each character you tell it to plot, and draws it to the screen as a series of horizontal lines. Since it has to draw eight of these lines, one at a time, for each character, and since it's in BASIC and needs all the speed it can get, we put it as close to the beginning of the program as possible.

Since the graphics characters are eight pixels (dots) high, you must use a 40 by 160 grid to position the first character of the U$ string. Also, make sure that U$ will fit on the screen. If it's too long, it will wrap around to the left side as usual, but to the next **pixel** down, not the next line (try it and see what happens).

G8PLOT does not support inverse letters, but prints all other characters. Graphics GOSUB routines will appear in the text window at the bottom of the screen only.

By the way (I almost forgot to mention it!), there is also a sample program as part of the G8PLOT file to let you see how it's done. After you feel comfortable, you can delete the sample code (lines 20 thru 150) and save what's left as a "GRAPHICS 8 Running Program".

HI THERE

The character set

The character set has been created to cover a wide range of applications and to be easy to remember.

A. **RECOGNITION OF KEY CODES**
Most of the special characters are printed to the screen by pressing the CTRL key and a letter key at the same time. Whenever possible, characters have been given letter assignments that are suggestive of the character (see the chart at the end of this section or type GOSUB KEYCODE and press RETURN if you have the Running Program loaded).

B. **SPECIAL TEXT CHARACTERS**
All upper and lower case keys print traditional renditions of the characters engraved on the keys with the following exceptions:

1.**Quotation Marks**
The "[" and "]" characters on the keyboard have been modified to produce quotation marks. They can be included inside print statements, making it possible to print quotations to the screen.

2. Mathematical Symbols
The multiplication and division symbols (* and /) have been modified to produce their more traditional representations (x and -).

The "/" symbol has been modified to print the degree symbol (" ") This makes it possible to print trigonometric functions to the screen in , ', and " (degrees, minutes, and seconds).

3. Escape Characters
The characters that use the ESC key are actually special commands for the operating system (see your Atari Reference Manual for their functions). These characters will invoke their designated commands instead of printing to the screen (even when included in print statements) unless they are preceeded by the ESC symbol (press the ESC key twice).

The subroutine GOSUB BORDER has been modified so the extra ESC symbol is not necessary in specifying escape characters in borders. Any colorful characters you print inside the quotation marks when specifying U$ will appear as border characters, whether they are escape characters or not.

The following table shows the decimal codes and keystrokes for all of the characters in YE OLDE TYPEFONT.

NEXT PAGE SWEETHEART

TYPEFONT PROGRAM

Ye Olde Typefont

Copyright 1982 by R. Wilson

CHARACTER CODES FOR CUSTOM CHARACTER SET - - [INVERTED (BLINKING) CHARACTERS]

DECOD.E	CHAR	FUNCTION KEYS	KEY
128		A C	,
129		A C	A
130		A C	B
131		A C	C
132		A C	D
133		A C	E
134		A C	F
135		A C	G
136		A C	H
137		A C	I
138		A C	J
139		A C	K
140		A C	L
141		A C	M
142		A C	N
143		A C	O
144		A C	P
145		A C	Q
146		A C	R
147		A C	S
148		A C	T
149		A C	U
150		A C	V
151		A C	W
152		A C	X
153		A C	Y
154		A C	Z
155	EOL	A	RETRN
156		ES	BKS
157		E S	>
158		EC	TAB
159		ES	TAB
160		A	SPACE
161	!	A S	!
162	"	A S	"
163	#	A S	#
164	$	A S	$
165	%	A S	%
166	&	A S	&
167	'	A S	'
168	(	A S	(
169	)	A S	)
170	*	A	*
171	+	A	+
172	,	A	,
173	-	A	-
174	.	A	.
175	/	A	/
176	0	A	0
177	1	A	1
178	2	A	2
179	3	A	3
180	4	A	4
181	5	A	5
182	6	A	6
183	7	A	7
184	8	A	8
185	9	A	9
186	:	A S	:
187	;	A	;
188	<	A	<
189	=	A	=
190	>	A	>
191	?	A S	?
192	@	S	@
193	A	A	A
194	B	A	B
195	C	A	C
196	D	A	D
197	E	A	E
198	F	A	F
199	G	A	G
200	H	A	H
201	I	A	I
202	J	A	J
203	K	A	K
204	L	A	L
205	M	A	M
206	N	A	N
207	O	A	O
208	P	A	P
209	Q	A	Q
210	R	A	R
211	S	A	S
212	T	A	T
213	U	A	U
214	V	A	V
215	W	A	W
216	X	A	X
217	Y	A	Y
218	Z	A	Z
219	[	A S	[
220	\	A S	\
221	]	A S	]
222	^	A S	^
223	_	A S	_
224	`	A C	.
225	a	A	a
226	b	A	b
227	c	A	c
228	d	A	d
229	e	A	e
230	f	A	f
231	g	A	g
232	h	A	h
233	i	A	i
234	j	A	j
235	k	A	k
236	l	A	l
237	m	A	m
238	n	A	n
239	o	A	o
240	p	A	p
241	q	A	q
242	r	A	r
243	s	A	s
244	t	A	t
245	u	A	u
246	v	A	v
247	w	A	w
248	x	A	x
249	y	A	y
250	z	A	z
251		A C	;
252		A S	=
253		EC	2
254		AEC	BKS
255		AEC	>
			=WHITE
			=YELLOW

TYPEFONT PROGRAM

Ye Olde Typefont

Copyright 1982 by R. Wilson

CHARACTER CODES FOR CUSTOM CHARACTER SET - - [KEYS - A=/|\ C=CTRL E=ESC S=SHIFT]

DEC.	CHAR	FUNCTION KEYS	KEY
0	--	C	,
1		C	A
2		C	B
3		C	C
4		C	D
5		C	E
6		C	F
7		C	G
8		C	H
9		C	I
10		C	J
11		C	K
12		C	L
13		C	M
14		C	N
15		C	O
16		C	P
17		C	Q
18		C	R
19		C	S
20		C	T
21		C	U
22		C	V
23		C	W
24		C	X
25		C	Y

DEC.	CHAR	FUNCTION KEYS	KEY
26		C	Z
27		E	ESC
28		EC	-
29		EC	=
30		EC	+
31		EC	x
32		SPACE	
33	!	S	!
34	"	S	"
35	#	S	#
36	$	S	$
37	%	S	%
38	&	S	&
39	'	S	'
40	(	S	(
41	)	S	)
42	x		x
43	+		+
44	,		,
45	-		-
46	.		.
47	÷		/
48	0		0
49	1		1
50	2		2
51	3		3

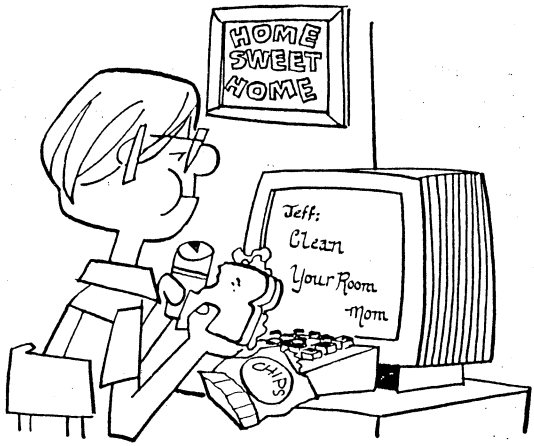
DEC.	CHAR	FUNCTION KEYS	KEY
52	4		4
53	5		5
54	6		6
55	7		7
56	8		8
57	9		9
58	:	S	:
59	;		;
60	<		<
61	=		=
62	>		>
63	?	S	?
64	@	S	@
65	A		A
66	B		B
67	C		C
68	D		D
69	E		E
70	F		F
71	G		G
72	H		H
73	I		I
74	J		J
75	K		K
76	L		L
77	M		M

DEC.	CHAR	FUNCTION KEYS	KEY
78	N		N
79	O		O
80	P		P
81	Q		Q
82	R		R
83	S		S
84	T		T
85	U		U
86	V		V
87	W		W
88	X		X
89	Y		Y
90	Z		Z
91	[	S	[
92	\	S	\
93	]	S	]
94	^	S	^
95	_	S	_
96	`	C	.
97	a		a
98	b		b
99	c		c
100	d		d
101	e		e
102	f		f
103	g		g

DEC.	CHAR	FUNCTION KEYS	KEY
104	h		h
105	i		i
106	j		j
107	k		k
108	l		l
109	m		m
110	n		n
111	o		o
112	p		p
113	q		q
114	r		r
115	s		s
116	t		t
117	u		u
118	v		v
119	w		w
120	x		x
121	y		y
122	z		z
123		C	;
124		S	=
125		E S	<
126		E	BKS
127		E	TAB
		=BLUE	
		=GREEN	

D. FOR MOMS AND DADS

With everyone working or in school, family messages and communications can get lost or forgotten. Use YE OLDE TYPEFONT as a family message center. When Jeff comes home from school and sees the TV repeating "Jeff -- clean your room!" (GOSUB GONG), he might even remember to do it. He might **even** be inspired to write you a reply, learning more about computers in the process (we won't take the blame for the reply, though!)



Technical Notes

A. BASIC VARIABLES USED IN THE RUNNING PROGRAM

To avoid unwanted interaction with the Running Program, simply do not use any variables in your program that start with the letter U (except U$, UB$, and GOSUB labels, which you obviously have to use). This is a completely safe approach.

For those reasonably familiar with BASIC, many of the variables **can** be used to advantage for memory conservation or expansion of the Running Program, and in that case the following information will be of interest.

All 35 variables used in the Running Program start with the letter U, except for the GOSUB labels.

TYPEFONT PROGRAM

Applications

HI THERE

A. FOR KIDS (and grown-up beginners)

This program allows the beginner to create professional quality graphics effects and fun sounds right away. It is ideal for simple question and answer games. The easy addition of colorful borders, humorous effects, and sounds provide an early reward that encourages the beginner to learn more.

B. FOR TEACHERS

YE OLDE TYPEFONT is ideal for classroom presentations of history, literature, poetry, etc. The colorful graphics, borders, and sounds allow you to add presentations to your classroom routine that hold attention and encourage imagination. It's also a welcome help and useful tool for substitute teachers. For science and math, the conventional symbols for multiplication, division, and degrees allow you to develop math drill games punctuated with interesting borders, gongs, and birds, all with a minimum of effort.

More advanced efforts include drawing geometric diagrams in traditional type. These can be supplemented with attention getting effects in the text window.

C. FOR BUSINESS APPLICATIONS

Video displays are proven winners in attracting customers. Even short presentations advertising your sales can be easily programmed to cycle endlessy in your window. With the Atari choosing an almost endless variety of random borders, even short presentations changed daily will attract customers. If you own a computer store, try cycling the demonstration program in you window and see what happens!

Visicalc (tm) and other such programs are great for intricate analysis of business problems. Many presentations, however, require summarizing the conclusion of such an analysis in such a way that people will be motivated. It may boil down to "Go get the competition!" (GOSUB GONG!). You can give your points a "hard sell" against an 1890's backdrop without seeming pushy or overly aggressive. Illustrate approximate trends with charts of little stacks of coins, etc. and get a laugh while you're at it.

TYPEFONT PROGRAM

1. VARIABLES USED AS CONSTANTS

	U0=0	U2=2	U6=10
U9=752			
	U1=1	U3=512	U8=256

All variables with number suffixes are used as constants. They are initialized to their specified value in line 32102. To conserve memory, you can use these variables to provide the same constant values in your program, but be sure not to change their values.

2. BASIC VARIABLES USED AS UTILITY VARIABLES

UA UB UC UD UN

These variables are always reinitialized before they are used in the Running Program, but they are left with indeterminate values after use. You can use them in your program without causing a malfunction of the Running Program, but be sure to use them only in applications where your program reinitializes them (such as an index for a FOR/NEXT loop), or your program will not work properly.

3. BASIC VARIABLES USED AS FLAGS

UM=1 border in effect UM=0 border not in effect

This variable should not be used in your program.

4. BASIC VARIABLES USED AS SUBROUTINE LABELS

There are 21 subroutine labels used in the Running Program. These labels are specified and assigned line number values in lines 32102 to 32104 of the program. A block of lines is left open so you can add your own subroutine calls. The subroutines themselves occupy lines 29980 to 30070. Lines 30082 to 30099 can be used for your subroutines. Care should be observed in renumbering the program, as GOSUB labels will have to have their values reassigned manually.

5. BASIC STRING VARIABLES

U$ (DIM 106) is a utility string that you, the user programmer, use to specify text and scroll messages in your program.

UB$ (DIM 6) is a utility string that you use to specify the characters in BORDERS when you wish to override the Atari's selection.

UA$ (DIM 34) is a data string containing the characters that the Atari uses to compose random borders.

6. MEMORY LOCATIONS USED AS VARIABLES

203-208: machine language variables (location 207 is blink control. 1 turns blink on, 0 turns it off).
1664: entry point for machine language subroutine to transfer data located at address specified in 1787 (low byte) and 1788 (high byte) to address specified in 205 (low byte) and 206 (high byte). POKE the source and destination addresses, then USR(1664). The subroutine returns to BASIC when it encounters CHR$(188) (reverse "<").
1702: start address of vertical blank blink subroutine.
1724: blink rate control (use values 0-4 only)
1772: entry point to initialize vertical blank subroutine.
1787: see 1664
1788: see 1664
1789: contains cursor row position +1 at completion of the 1st GOSUB
1790: start address (high byte) of the custom character set.
1791: 0=running program not initialized, 1=program initialized.

7. OPERATING SYSTEM VARIABLES USED

The Running Program uses PEEKs and POKEs to Operating System variables that have the functions described below.

84: current cursor row position in GR.0 (ROWCRS)
85,86: current cursor column position in GR.0 (COLCRS)
87: graphics mode (DINDEX)
186,187: line number triggering error trap (STOPLIN)
656: current cursor row position in text window (TXTROW)
752: cursor visible/invisible (CRSINH)
764: internal code for last key pressed (CH)
755: character mode register (CHAR)
756: high byte of address of Atari character set (224=CAPS) (CHBAS)

B. MEMORY ALLOCATION

The Running Program uses the upper half of page six (1664-1791) for machine language subroutines and variables not affected by SYSTEM RESET. Page six memory locations 1536 to 1663 and 1782 to 1786 are unused and available for your use.

The machine language subroutines use page zero addresses 203 to 208. 209 is not used and 203, 204, and 208 are reinitialized by the machine language subroutines.

The character set is located at RAMTOP-8, and the 2048 bytes from this location to the end of the display list are needed to assure non-interference with the Operating System. This allocation is specified in line 32118 of the Running Program.

The BASIC Running Program occupies 7030 bytes and is located in memory by the Operating System. This configuration makes the program compatible with both cassette and disk operation.

The character set is contained in the BASIC program as DATA statements encoded in ATASCII code, in lines 32300 to 32310. This compacts the character set and makes it easy to save your own programs in BASIC utilizing the character set with a minimum load time if you're using cassette.

The Running Program is purposely not interlocked to prevent your program from writing over the character set. The vital parts of the Running Program are protected in the upper half of page six.

If your program begins to exceed available memory, the graphics and lower case letters in the typefont will start to look funny. However, the characters used in programming will still look alright and you can program yourself out of the situation instead of having the Atari "lock up".

Note that inquiries of ? FRE(0) will not allow for the 2K used for the character set. The FRE(0) reported by the "Something is Amiss" error trap does allow for this. When this value is a negative number, it is to tell you the number of bytes of overlap with the character set.

C. GENERAL OPERATION OF THE RUNNING PROGRAM

Lines 0 and 1 contain the SOUND statement for the bird call subroutine. This is ignored if not called for. Line 2 directs the program to the initialization routine at 32100 if the program is not properly initialized.

The first time through, 1791 has a value of zero and the initialization routine POKEs the short machine language routine (line 32110) for rapid relocation of data into page six. This routine in turn is used to set up the rest of page six with the blink subroutine, and to transfer the character set from BASIC DATA statements to its proper location referenced eight pages below the RAMTOP of your particular Atari.

The program then stores the beginning address of the character set in 1790 and POKEs the initialization flag (1791) with a one (line 32122). It then goes to line 29905, where it gives a signal that the program is initialized. It then jumps to 29985.

This section of the program resides at the end of the program you write. It keeps program control until you either press OPTION or any other key. If you press OPTION, it will list your portion of the program only and remain under program control until some other key is pressed. If some other key is pressed, the program reverts to BASIC, utilizing the keystroke as it normally would.

When you run **your** program, the Running Program (line 2) checks to see whether critical variables have been disturbed by the operating system. If either variables or the character set have been disturbed, the program is diverted to the first part of the initialization process where they are restored. The Running Program then jumps to execute your program. If critical variables are not disturbed, the above process is bypassed and your program is executed directly.

After executing your program, the Running Program again goes to 29985, where it maintains program control until you press OPTION or any other key.

The error message routine is on lines 32000 to 32010, and the rest of the program is devoted to the subroutines (addressed on lines 32102 to 32104) that support the program.

A short subroutine to display the special characters resides at the end of the program after the data statements, and can be removed without ill effect once you become familiar with the character set.

D. MODIFYING THE RUNNING PROGRAM

You may wish to modify the Running Program to include special sound subroutines you have collected or other routines you want to use repetitively.

To do this, put your subroutine in lines 30082 to 30099 or at the end of the program. Then add your subroutine variable name (equated to the line number where the subroutine starts) or a line number in the 32105 to 32107 range (reserved for this purpose). The program contains remark statements in inverse letters to help you locate these areas when listing your program.

If you run short of memory, you can delete remark statements and the display of the character set at the end of the program.

If you would like the Atari to say something other than "Something is Amiss" in response to an error, then by all means change the message where it appears in line 32000.

If you would like characters other than a blue ball to appear in the GOSUB BALL command, then change the statement "- O"; in line 30029 to "- ";CHR$(uc). Now you can specify UC before you GOSUB BALL. For example:

```
UC=10:U$="KISS":GOSUB BALL
```

in your program would then produce a heart moving to a scroll containing the word "KISS".

You can get better (but more complicated) control of program timing by making the following changes:

1. Add SPEED=U to line 32103
2. Change line 30056 to the following:

```
30056 FOR UN=U0 TO INT(UT/SPEED):UC=U1*U6:NEXT
      UN:RETURN
```

Now you can specify an individual delay in your program by specifying UT (four counts to a second) before each GOSUB TIME call. If you use this method for all delays in your program, then you can alter the speed of the presentation by specifying different values for SPEED (SPEED=1 is normal, SPEED=1.1 is 10% faster, SPEED=.9 is 10% slower).

Don't hesitate to make these or other alterations to the program in order to have it perform the way you want it to.

FX-80 DOWNLOADER

Of course, although it's always nice to have snazzy fonts on the screen, it's even nicer to have them on paper. Unfortunately, not all printers allow you to "download" a custom font. However, if you own an Epson FX-80 printer, FX80DOWN will allow you to print using your own custom-designed font. Use CHAREDIT to design your font first, and then let FX80DOWN do the rest (FX80DOWN is a bonus for disk users only). We could not provide a similiar program for other less popular printers because each printer works differently and each must be in machine language.

LOADING FX80DOWN

Turn your Atari off and remove any cartridges. Make sure your printer is on (and interface if necessary). Turn on your disk drive and insert the FANCY FONTS diskette. Turn on your Atari and wait for DOS to appear on the screen. Choose option 'L' (for binary load) and type 'FX80DOWN' as the file to be loaded from. The program will then load and run automatically.

USING FX80DOWN

Using this program is incredibly simple. If you want to download a font that is stored on diskette, just type the filename (don't forget to specify D:) in response to the prompt;

Load font from what file?

and press RETURN. If you want to download the internal character set (the font that the Atari normally uses), then press RETURN without typing a filename.

If the Atari has any problems loading in your font from disk, it will tell you:

DISK ERROR-TRY AGAIN

If this happens, double check that you typed the filename correctly, the correct diskette is in the disk drive, the drive door is closed, and all the other wonderful disk things that could be causing a problem.

THIS IS SOME FUN

FX-80 DOWNLOADER

Once the font is loaded into memory, you will see:

THINKING...

on the screen, with a countdown next to it. When the countdown reaches zero (it takes about a minute), the Atari will then start

DUMPING...

the font to the printer. This only takes a few seconds, after which

OPTION TO BOOT
SELECT FOR DOS
START TO RERUN

will appear on the screen. Congratulations! Your Epson now has a custom font to print with. This font will remain in the printer until the **printer** is turned off or another font is downloaded. It will **not** disappear when the computer is turned off. This means that you can load in your favorite word processor and print documents using the new font.

Incidentally, regarding those last options to appear on the screen, pressing the OPTION button will let you reboot the Atari (that's the same as turning the computer off and then on again), the SELECT button will return you to DOS, and the START button will run the program over in case you made a mistake (these three options are a result of the fact that FX80DOWN was produced using copyrighted software products of Monarch Data Systems, Cochituate, MA 01778. In other words, it was compiled from BASIC to make it run faster).

PROBLEMS

If you've already followed the above instructions and have tried printing with your new font, you probably noticed that the letters don't look quite as good as they did on the screen, and some may be plain unreadable. Don't worry - the program is working ok. What you are seeing is the result of the way the Epson is designed; it won't print two dots side by side. If you have two dots side by side in a character, the second one won't be printed. If you have three side by side, the second one won't be printed but the first **and** third will (if there was a fourth, the fourth wouldn't be printed and so on). What this means is that you'll probably want to make changes to existing fonts to make sure they look ok on the Epson, or even custom design fonts especially

for the Epson. Another thing you should keep in mind when adjusting fonts for the Epson is to make them wider. The Atari needs some space in each character so that characters side by side won't touch. This is automatically taken care of by FX80DOWN, so you can, and should, make characters a full eight dots wide. Finally, Atari characters need vertical lines to be two dots thick. The Epson doesn't and in fact, as I explained above, it messes things up. As a matter of fact, this is the biggest reason that the characters do look funny, so don't make things thicker than they have to be!

TECHNICAL STUFF

If FX80DOWN doesn't work at all for you, it's probably because of a little switch inside the Epson called DIP SW1-4. In order for downloaded fonts to work, this switch must be set to 'off'. It was set to 'off' at the factory, but somebody might have messed with it since then so get out your Epson manual, look up how to set the switches, and make sure that this one is set to 'off' (it's real easy to do this).

LOTS OF FONTS .....

If you look at the table of contents, you will see lots of fonts already prepared for you. You can use the character editor to load and look at each one. Here are a few lines printed on the FX80 to show you some of the possibilities:

This is written using COMPUTER FONT !

This is written using STYLISH Font !

This is written using MATH Font !

From this point on it's all up to you. These tools can create some wild displays. Send us your best efforts and maybe we will include a few in later versions of this or other tutorials. BYE!

SUMMARY OF COMMANDS

GOSUB ATARI

1. Makes all characters on the screen appear in the normal Atari typefont.
2. Does NOT change the color of the screen.

GOSUB BALL

1. Moves the cursor one line down, justified to the left side of the picture.
2. Draws a hand, then a ball moving right to a SCROLL which is right justified on the scree.
3. The message in the scroll contains the characters specified in U$ (see GOSUB SCROLL)

GOSUB BIRD

1. Makes a bird chirping sound.
2. Moves the cursor down one line and prints hearts & flowere centered on tha screen line.
3. Moves the cursor one line down justified left.

GOSUB BORDER

1. Clears the screen.
2. Draws a border with the characters specified in UB$ (random selection if you don't specify UB$).
3. Places the cursor so the next character will be in the upper left, justified inside the border.
4. Sets a "flag" so that GOSUB commands that follow will justify inside the border.

EXAMPLE: 10 UB$="ABCD"
20 GOSUB BORDER

Note: Horizontal borders will repeat the pattern ABCD and vertical borders will repeat the pattern AB

5. Use the command GOSUB EXIT when finished with a presentation inside a border. (This puts the cursor where it won't spoil the picture, and resets the border flag).

GOSUB BON

1. Makes inverted characterss on the screen blink. (The start button also turns the blinking on).

GOSUB BOFF

1. Turns blinking characters off. (The start button also turns the blinking off).

GOSUB BTIME

1. Makes inverted characters blink for a few seconds and then stop).

GOSUB GONG

1. Rings a 'Gong'. Program delays until the gong sound is complete.

GOSUB HAND

1. Draws a pointing hand on the screen, starting at the last cursor position.
2. Leaves the cursor one position to the right of the fingertip.
3. GOSUB CHAND clears the screen and then draws a hand justified at the upper left of the screen.

SUMMARY OF COMMANDS

GOSUB KEYCODE

1. Displays the special characters in "YE OLDE TYPEFONT" and the keystrokes to print them.

GOSUB OLDTYPE

1. Makes all characters on the screen appear in "Ye Olde Typefont".
2. Sets background color on the screen (and text window) to black.

GOSUB SCROLL

1. Starts from the current cursor position and draws a SCROLL, enclosing the message in U$.

EXAMPLE: 10 U$="ARMYBOOTS":GOSUB:SCROLL:REM DRAWS A SCROLL AROUND THE WORD 'ARMYBOOTS'

2. Moves the cursor two lines below the text in the scroll, justified on the left of the screen.
3. GOSUB MSCROLL centers the scroll in the middle of the screen on the current line.
4. GOSUB CMSCROLL clears the screen and draws the scroll in the middle of the top screen line.
5. Specify INVERSE characters in U$ for blinking scrolls.
6. If the scroll says 'too long' then shorten the message in U$ so it fits on one screen line.

GOSUB SWTYPE

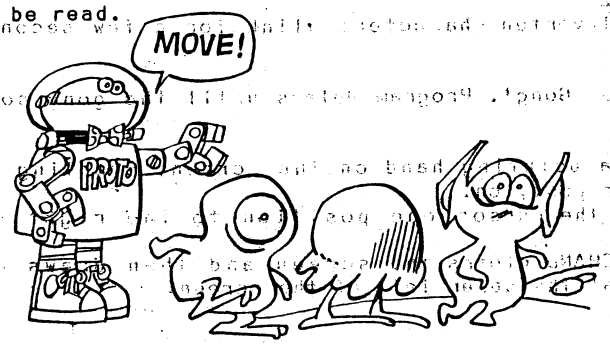
1. Switches all characters on the screen back and forth from normal Atari characters to "Ye Olde Typefont" characters for several seconds.

GOSUB TEXT

1. Left justifies up to three lines of text specified in U$ (full screen or within border).
2. Leaves the cursor positioned one space past the end of the text message printed to the screen.
3. If text is inside a border & the presentation exceeds one screen frame, the supporting program will automatically display a completed frame & draw the border for the next screen.
4. MTEXT centers one screen line of text. CMTEXT clears the screen & centers one line of text.

GOSUB TIME

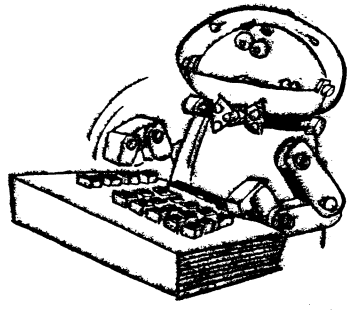
1. Freezes execution of the program for five seconds so the display can be read.



TYPEFONT PROGRAM

Note: TYPEFONT.PGM = 0-2 AND 29984-32327
TYPEFONT.DMO = ALL OF LISTING

```
0 IF UB=1 THEN FOR UC=U1 TO 3
    :FOR UN=25 TO U0 STEP -U1
    :SOUND U0,UN,U6,14
    :NEXT UN
    :? "[illegible]";
    :NEXT UC
    :FOR UC=U4 TO 0
1 IF UB=1 THEN FOR UN=U0 TO U6
    :SOUND U0,UN,U6,U6
    :NEXT UN
    :? "[illegible]";
    :NEXT UC
    :UB=U0
    :SOUND U0,U0,U0,U0
    :RETURN
2 TRAP 32000
    :IF U0=U1 THEN 32100
    :REM VE OLDE TYPEFONT-REV.0.0-Copyright
    )1982by R Wilson [illegible]
[illegible]
4 FOR A=0 TO 14
5 FOR N=0 TO 14
    :? "SETCOLOR^4,";A;",";N;
    :SETCOLOR 4,A,N
    :UA=U1^3
    :UA=U1^3
    :UA=U1^3
    :UA=U1^3
    :NEXT N
6 NEXT A
10 K=6
    :SETCOLOR 4,4,U0
    :POKE U9,U1
20 ? "[illegible]"
30 U$="[illegible]"
    :? U$
40 ? "[illegible]"
50 ? "[illegible]"
60 ? "[illegible]"
70 ? "[illegible]"
80 ? "[illegible]"
90 ? "[illegible]"
100 ? "[illegible]"
110 ? U$
120 ? "[illegible]Copyright^1982^by[illegible]"
130 ? "[illegible]Rich^Wilson[illegible]"
140 ? U$
150 ? "[illegible]ABCDEFGHIJKLMNOPQRSTUVWXYZ[illegible]"
160 ? "[illegible]"
170 ? "[illegible]abcdefghijklmnopqrstuvwxyz[illegible]"
180 ? "[illegible]"
190 ? "[illegible]Pilgrims^1620[illegible]178\^West[illegible]"
200 ? "[illegible]"
210 ? "[illegible]Coins[illegible]"
220 ? "[illegible]Cyphers[illegible]180/60=3[illegible]"
230 ? U$
240 ? "[illegible]"
245 REM REV. 0.0 3-2-83
250 GOSUB TIME
    :GOSUB BTIME
260 U6$="[illegible]"
    :GOSUB BORDER
    :?
    :?
    :U$="[illegible]"
    :GOSUB MSCROLL
    :?
    :GOSUB BIRD
    :?
    :GOSUB HAND
270 U$="Use^TYE^OLDE^TYPEFONT)^to^create^progr[illegible]^and^messages^"
    :GOSUB TEXT
    :? "LE"
    :GOSUB BIRD
    :?
    :U$="[illegible]"
280 GOSUB BALL
    :GOSUB SONG
    :GOSUB EXIT
290 FOR N=U0 TO U8
    :NEXT N
    :GRAPHICS U2
    :GOSUB OLDTYPE
    :POSITION 4,4
    :? #6;"WRITE^POETRY"
    :U$="[illegible]"
    :GOSUB BALL
300 GOSUB BTIME

    :GRAPHICS U0
    :GOSUB OLDTYPE
310 SETCOLOR 4,15,U0
    :U8$="[illegible]"
    :GOSUB BORDER
320 U$="[illegible]"
    :GOSUB MSCROLL
    :? "f";
    :GOSUB HAND
330 U$="Let^dogs^delight^to^bark^&^bite,
    ^for^god^hath^made^them^so;"
```



```
    :GOSUB TEXT
    :?
    )?
340 U$="Let^bears^and^lions^growl^and^
    fight,^for^'tis^their^nature^to."
    :GOSUB TEXT
    :?
    :?
    :?
350 U$="But,^children,^you^should^never^
    let^such^angry^passions^rise,"
```

TYPEFONT PROGRAM

```
    :GOSUB TEXT
    :?
    :?
360 U$="Your~little~hands~were~never
    ~made~to~tear~each~other's~eyes."
    :GOSUB TEXT
    :?
    :?
    :GOSUB BIRD
    :?
370 U$="ISAAC~WATTS...........(1674-1748)"
    :GOSUB TEXT
    :GOSUB BTIME
    :GOSUB EXIT
    :GOSUB TIME
380 GRAPHICS U2
    :GOSUB OLDTYPE
    :POSITION 7,3
    :? #K;"AMAZE"
    :GOSUB SWTYPE
    :? #K
    :? #K;"~~~~~YOUR~BOSS"
390 U$="[illegible]"
    :GOSUB BALL
    :GOSUB BTIME
    :GRAPHICS U0
    :GOSUB OLDTYPE
400 GOSUB BORDER
    :?
    :?
    :U$="Dear~Mr.~President:"
    :GOSUB TEXT
    :?
    :?
    :U$="This~month's~report~is........"
410 GOSUB TEXT
    :?
    :? "→";
    :U$="[illegible]"
    :GOSUB BALL
    :GOSUB BIRD
    :? "↓↓→→→↓→~~";
420 U$="I~[illegible]~have~led~the~[illegible]
    [illegible]~to~unexcelled~perfo
rmance,~so~that←←←←←←←←←↓→"
    :GOSUB TEXT
430 GOSUB EXIT
    :GOSUB BTIME
440 GRAPHICS U2
    :GOSUB OLDTYPE
    :POSITION K,3
    :? #K;"PROFITS"
    :? #K
    :? #K;"~~~AT~WIDGET~CORP"
    :GOSUB SWTYPE
450 U$="[illegible]"
    :GOSUB BALL
    :GOSUB GONG
    :GRAPHICS U0
    :GOSUB OLDTYPE
460 GOSUB BORDER
    :U$="[illegible]"
    :GOSUB MSCROLL
    :RESTORE
    :FOR N=U1 TO 12
    :READ A
    :IF A>S THEN S=A
470 NEXT N
    :POKE 752,U1
    :S=U6/S
    :POSITION 5,5
    :U$="$~MILLIONS"
    :FOR N=U1 TO U6
    :? U$(N,N);"↓←";
    :NEXT N
    :POSITION 7,5
480 FOR N=U6 TO U0 STEP -U2
    :? INT(N*S);"_↓↓←↓←←←↓";
    :NEXT N
    :? "↑→→~↑←_";
    :FOR N=U1 TO 24
    :? "_";
    :NEXT N
490 DIM M$(36)
    :M$="JanFebMarAprMayJunJulAugSepOctNovDec"
    :POSITION 11,16
    :B=U1
500 FOR N=B TO B+33 STEP 3
    :? M$(N,N);"~~";
    :NEXT N
    :POSITION 11,PEEK(84)+U1
    :B=B+U1
    :IF B<4 THEN 500
510 X=11
    :RESTORE
520 READ A
    :POKE U0,A*S
    :A=PEEK(U0)
    :Y=15-A
    :POSITION X,Y
    :IF Y=15 THEN ? "↑_"
    :GOTO 540
530 FOR N=Y TO 14
    :? "↓↓←";
    :NEXT N
540 X=X+U2
    :IF X<34 THEN 520
550 ? "[illegible]";
    :GOSUB BTIME
    :GOSUB EXIT
560 DATA 6,2,4,7,4,5,6,5,4,7,6,10
570 REM [illegible]
    [illegible]
580 GOSUB BORDER
    :? "↓↓"
    :U$="Respectfully~submitted;"
    :GOSUB MTEXT
    :?
    :?
590 U$="[illegible]"
    :GOSUB MSCROLL
    :GOSUB BIRD
    :? "↓↓"
    :U$="DIVISION~MANAGER"
    :GOSUB HAND
600 GOSUB MSCROLL
    :GOSUB GONG
    :GOSUB BTIME
    :GOSUB EXIT
610 GRAPHICS U2
    :GOSUB OLDTYPE
    :POSITION 8,U2
    :SETCOLOR U2,4,U0
    :COLOR 3
    :? #K;"draw"
    :POSITION 3,4
    :? #K;"NIFTY~PICTURES"
620 U$="[illegible]"
    :GOSUB BALL
    :GOSUB GONG
    :GOSUB BTIME
```

TYPEFONT PROGRAM

```
630 GRAPHICS U0
    :COLOR U1
    :GOSUB OLDTYPE
    :GOSUB BORDER
    :POSITION 17,K
    :? "[illegible]"
640 SETCOLOR 4,3,U0
650 POSITION 16,9
    :? "[illegible]"
    [illegible]
    :U$="[illegible]"
    :GOSUB HAND
    :GOSUB SCROLL
660 GOSUB BTIME
    :POSITION 18,8
    :POKE U9,U1
    :? "-";
    :A=K^K
    :A=K^K
    :A=K^K
    :A=K^K
    :? "[illegible]"
    :POSITION 4,18
    :GOSUB BIRD
670 GOSUB EXIT
    :POKE U9,U0
680 GRAPHICS U2
    :GOSUB OLDTYPE
    :POSITION K,3
    :? #K;"stupify"
    :GOSUB SWTYPE
    :GOSUB ATARI
    :? #K
690 ? #K;"~HUGE~COMPUTER~BANKS"
    :GOSUB GONG
    :? #K;"~~~~~WITH~YOUR"
    :U$="[illegible]"
    :GOSUB OLDTYPE
    :GOSUB BALL
700 GOSUB BTIME
    :GRAPHICS U0
    :GOSUB OLDTYPE
    :GOSUB BIRD
    :?
    :? "Code#~ANP-760838+452-A9....
 Charge~to~DADDY~WARBUCKS"
710 ?
    :? "To~[THE~SAUCE]"
    :? "Division~of~RODDERS~DIGEST"
    :?
    :? "Dear~SAUCE:"
    :?
    :? "Your~latest~Newsflash~Transmis";
720 ? "sion~~~~about~the~KUMQUAT~harvest~
    [illegible]"
    :SETCOLOR 4,3,4
    :U$="[illegible]"
    :GOSUB BALL
    :? "↑";
    :GOSUB GONG
730 GOSUB BIRD
    :LIST 32306,32307
    :? "↑↑";
    :POKE 752,U1
    :GOSUB BTIME
    :GOSUB BTIME
    :GOSUB BTIME
    :GRAPHICS U2
740 GOSUB OLDTYPE
    :POSITION 3,U2
    :? #K;"LEAVE~MESSAGES"
    :GOSUB HAND
    :GOSUB BIRD
    :? #K
    :? #K;"~~~FOR~THE~FAMILY"
750 POKE U9,U1
    :?
    :GOSUB BTIME
    :GRAPHICS U0
    :GOSUB OLDTYPE
    :GOSUB BORDER
760 ?
    :U$="~Dear~Dad~"
    :GOSUB MSCROLL
    :GOSUB BIRD
    :?
    :U$="I~Borrowed~the~car~for~a~while.
          ~~I'll~be~back~in.....~"
770 GOSUB TEXT
    :GOSUB BTIME
    :U$="[illegible]"
780 GOSUB BALL
    :GOSUB GONG
    :GOSUB BIRD
    :? "↓↓"
    :U$="Your~son,~Jeff"
    :GOSUB MTEXT
790 GOSUB BTIME
    :GOSUB EXIT
800 GRAPHICS U2
    :GOSUB OLDTYPE
    :SETCOLOR 4,4,U0
    :POSITION U2,4
    :? #K;"$~CASH~CUSTOMERS"
    :GOSUB GONG
    :POSITION K,U2
810 ? #K;"attract"
    :POSITION 3,K
    :? #K;"TO~your~STORE"
    :GOSUB SWTYPE
    :POSITION 7,8
    :? #K;"WITH"
    :U$="[illegible]"
820 GOSUB BALL
    :GOSUB GONG
    :GOSUB BTIME
    :? #K;"↑"
    :U$="~HARRY'S~HARDWARE~"
    :GOSUB BALL
    :GOSUB GONG
830 POSITION U1,4
    :FOR N=U1 TO U2
    :? #K;"save~";
    :GOSUB GONG
    :SETCOLOR U2,K,U0
    :? #K;"SALE~";
    :GOSUB GONG
840 SETCOLOR U2,12,U0
```

```
    :NEXT N
    :POKE U9,U1
    :? "¶↓~~~~~~~~~~$~SAVE~AT~HARRY'S~$"
    :GOSUB SWTYPE
    :GRAPHICS U0
 850 GOSUB OLDTYPE
    :UB$="[illegible]"
    :GOSUB BORDER
    :?
    :GOSUB MSCROLL
    :GOSUB BIRD
    :U$="[illegible]"
    :GOSUB MTEXT
 860 GOSUB BIRD
    :?
    :U$="¬~HAMMERS~¬~HAMMERS~¬~HAMMERS~¬
    :GOSUB MTEXT
    :GOSUB BTIME
    :?
    :GOSUB BIRD
    :U$="~BUY~10~"
    :?
 870 GOSUB MSCROLL
    :? "↑";
    :U$="AND~GET"
    :GOSUB MTEXT
    :U$="[illegible]"
    :GOSUB BALL
    :GOSUB GONG
    :GOSUB BTIME
    :GOSUB EXIT
 880 GOSUB BORDER
    :? "↓↓↓"
    :U$="[illegible]"
    :GOSUB MSCROLL
 890 U$="Press~[illegible]~several~times~to~see~how
    ~to~activate~blinking~charac
ters~~manually.~~Check~page~3,~paragraph~"
900 GOSUB TEXT
    :U$="C~in~your~instruction~book~for~some
~simple~commands~that~~you~c
an~try~at~this~point."
910 GOSUB TEXT
    :GOSUB EXIT
29984 REM [illegible]
    [illegible]
    [illegible]
29985 POSITION U2,PEEK(84)
    :? "↓READY"
    :U$="[illegible]"
29986 IF PEEK(53279)=5 THEN ?
    :?
    :?
    :?
    :? "↑↑↑↑↑"
    :GOSUB BALL
    :LIST 3,29983
    :UC=U3
29987 IF UC=U3 THEN FOR UN=U0 TO 99
    :NEXT UN
    :?
    :?
    :?
    :?
    :?"↑↑↑↑↑"
    :UC=U0
    :GOTO 29985
29988 IF PEEK(764)=255 THEN 29986
29989 POSITION U2,PEEK(84)-U2
    :U$="SPECIFY~U$"
    :GOSUB BOFF
    :END
29990 ? "¶";
29992 UB=85
    :IF PEEK(87)<>U0 THEN UB=657
29993 IF LEN(U$)>36-4*UM THEN U$="U$~IS~TOO~LONG"
29994 POKE UB,(40-LEN(U$))*0.5
    :UD=U1
    :GOTO 30000
30000 UC=U2
    :UN=36
    :IF UM=U1 THEN UC=4
    :UN=33
    :IF PEEK(85)<UC THEN POKE 85,UC
30002 UA=U1
    :UB=UC+UN-PEEK(85)
    :POKE U9,U1
30004 IF UA>LEN(U$) THEN POKE 85,PEEK(1789)
    :POKE 752,U0
    :IF UD=U1 THEN ?
    :UD=U0
30005 IF UA>LEN(U$) THEN RETURN
30006 IF U$(UA,UA)="~" THEN UA=UA+U1
    :UB=UB+U1
    :GOTO 30004
30008 IF UB>=LEN(U$) THEN UB=LEN(U$)
    :GOTO 30016
30010 IF U$(UB,UB)<>"~" AND U$(UB+U1,UB+U1)<>"~"
    THEN UB=UB-U1
    30012 IF UB<UA THEN UB=UA+UN
        :GOTO 30018
    30014 IF U$(UB,UB)<>"~" THEN UB=UB-U1
        :GOTO 30012
    30016 ? U$(UA,UB);
        :UA=UB+U1
        :UB=UA+UN
        :POKE 1789,PEEK(85)+U1
        :IF UA>LEN(U$) THEN 30004
    30018 IF PEEK(84)>19 THEN GOSUB BTIME
        :GOSUB TIME
        :? "¶";
        :IF UM=U1 THEN GOSUB BORDER
    30020 POSITION UC,PEEK(84)+U1
        :GOTO 30004
    30022 IF UB$="~~~~~" THEN GOSUB 32311
    30023 ? "¶";
        :UM=U1
        :POKE U9,U1
    30024 POSITION U2,U0
        :FOR UN=U1 TO 9
        :? "%";UB$(U1,U1);"%";UB$(U2,U2);"%";UB$
        (3,3);"%";UB$(4,4);
        :NEXT UN
    30025 FOR UN=U1 TO 11
        :? "%";UB$(U1,U1);"↓+";"%";UB$(U2,U2);"↓+";
        :NEXT UN
        :POSITION U2,U0
        :FOR UN=U1 TO 11
    30026 ? "%";UB$(U1,U1)
        :? "%";UB$(U2,U2)
        :NEXT UN
        :FOR UN=U1 TO 9
        :? "%";UB$(U1,U1);"%";UB$(U2,U2);"%";UB$
        (3,3);
    30027 ? "%";UB$(4,4);
        :NEXT UN
        :? "%";UB$(U1,U1)
        :POSITION 3,U2
        :? "~";
        :POKE U9,U0
        :RETURN
    30028 ?
        :POKE U9,U1
        :IF PEEK(84)>20 THEN ?
        :?
        :? "↑↑↑"
```

```
30029 GOSUB HAND
    :FOR UN=U1 TO 24-LEN(U$)-UM-UM
    :? "←^↑";
    :UC=U1/U3/U3
    :NEXT UN
    :? "←";
    :GOSUB SCROLL
    :RETURN
30030 ? "↑";
    :UM=U0
30032 IF UM=U1 THEN ? "→";
30033 IF PEEK(87)<>U0 THEN IF PEEK(656)=U0
      OR PEEK(656)=3 THEN POKE 656,
U1
30034 ? "→◢└┘↓↓↓↓↓↓↑↑↑↑↑↑↑↑.●^";
    :RETURN
30035 FOR UN=13 TO U1 STEP -U1
    :SOUND U0,U6,U2,UN
    :UC=U1^U8
    :NEXT UN
    :SOUND U0,U0,U0,U0
    :RETURN
30036 ? "↑";
30037 UB=85
    :IF PEEK(87)<>U0 THEN UB=657
30038 POKE UB,(38-LEN(U$))*0.5
    :GOTO 30042
30040 ? "↑";
30041 IF UM=U1 AND PEEK(85)<4 THEN ?
      "→→→→";
30042 IF LEN(U$)>(31-PEEK(85)) THEN
      U$="U$^TOO^LONG"
30043 IF PEEK(84)>22 THEN ?
    :? "↑";
30044 IF PEEK(84)=U0 THEN ? "↓";
30045 IF UM=U1 AND PEEK(84)<U2 THEN
      POKE 84,U2
30046 IF PEEK(87)<>U0 THEN IF
  PEEK(656)=U0 OR PEEK(656)=3 THEN POKE 656,
  U2
30047 POKE U9,U1
    :? "↓↑↑↑↑↑";
    :FOR UN=U1 TO LEN(U$)+U2
    :? "=";
    :NEXT UN
    :? "↓↑↑↑↑↑↓↑↑↑";
    :FOR UN=U1 TO LEN(U$)+U2
    :? "←←";
30048 NEXT UN
    :? "↑→→";U$(U1,LEN(U$))
    :?
    :IF PEEK(87)=U0 THEN ?
30049 POKE U9,U0
    :RETURN
30054 POSITION U2,21
    :UM=U0
    :UB$="^^^^^^"
    :RETURN
30056 FOR UN=U0 TO 20
    :UC=U1^U8
    :NEXT UN
    :RETURN
30058 POKE 207,U1
    :RETURN
30060 POKE 207,U0
    :POKE 755,U2
    :RETURN
30062 POKE 207,U1
    :GOSUB TIME
    :GOSUB BOFF
    :RETURN
30064 POKE U9,U1
    :POSITION 9,PEEK(84)
    :UB=U1
    :GOSUB U0
    :?
    :POKE U9,U0
    :RETURN
30066 SETCOLOR U2,U0,U0
    :POKE 756,PEEK(1790)
    :RETURN
30068 POKE 756,224
    :RETURN
30070 FOR UC=U0 TO 20
    :POKE 756,224
    :FOR UN=U0 TO U6
    :NEXT UN
    :GOSUB OLDTYPE
    :FOR UN=U0 TO 15
    :NEXT UN
    :NEXT UC
    :RETURN
31000 GOSUB BOFF
    :GOSUB CHAND
    :GOSUB BIRD
    :GOTO 29985
32000 GOSUB CHAND
    :U$="[illegible]"
    :GOSUB SCROLL
    :GOSUB BIRD
32002 UN=PEEK(186)+U8*PEEK(187)
    :UC=PEEK(195)
32004 ?
    :? "|||It's^a^NO-NO^TYPE^[illegible]";UC;"^on ?
    THIS^line."
    :? "[illegible]↑"
32007 IF UN=29999 THEN ?
    :? "KEYBOARD^INPUT"
    :GOTO 32010
32008 LIST UN
    :?
    :? "N=";N;"^I=";I;"^J=";J;"^A=";A;
    "^X=";X;"^Y=";Y;"^^FRE(U0)=";FRE(U
0)-1024
32010 U$="SPECIFY^U$"
    :GOSUB BTIME
    :GOTO 29985
32100 DIM U$(106)
    :DIM UA$(34)
    :DIM UB$(6)
    :UB$="┐┌┐┌"
    :U$="SPECIFY^U$"
    :IF PEEK(87)>3 THEN GRAPHICS U0
    :POKE 1791,U0
32102 ? "↑";
    :U1=1
    :U2=2
    :U8=256
    :U3=U8+U8
    :U6=10
```

```
    :U9=752
    :SETCOLOR U2,U0,U0
    :TEXT=30000
    :BORDER=30022

    :BALL=30028
32103 CHAND=30030
    :HAND=30032
    :GONG=30035
    :CMSCROLL=30036
    :MSCROLL=30037
    :SCROLL=30042
    :EXIT=30054
    :CMTEXT=29990
32104 TIME=30056
    :BON=30058
    :BOFF=30060
    :BTIME=30062
    :BIRD=30064
    :OLDTYPE=30066
    :ATARI=30068
    :SWTYPE=30070
    :MTEXT=29992
32105 KEYCODE=32321
32108 IF PEEK(1791)=U1 THEN GOSUB OLDTYPE
    :UN=USR(1772)
    :GOSUB BOFF
    :? "¶";
    :GOTO 2
32110 U$(U1,38)="h[illegible]
    [illegible]"
    :UA=ADR(U$)
    :FOR UN=U0 TO 38
    :POKE 1664+UN,PEEK(UA+UN)
32112 NEXT UN
    :UB=UA-U1
    :UC=INT(UB/U8)
    :POKE 1788,UC
    :POKE 1787,UB-UC*U8
32114 POKE 206,6
    :POKE 205,166
    :RESTORE 32200
32116 READ U$
    :UA=USR(1664)
    :IF U$(81,81)<>"[illegible]" THEN 32116
32118 UA=PEEK(106)
    :UB=U8*(UA-8)
    :POKE 206,UA-8
    :POKE 205,U0
    :UC=UB-U1
    :RESTORE 32300
32120 READ U$
    :UA=USR(1664)
    :IF U$(25,25)<>"[illegible]" THEN 32120
32122 UN=UB/U8
    :POKE 756,UN
    :POKE 1790,UN
    :UA=USR(1772)
    :POKE 1791,U1
    :RESTORE
    :GOTO 31000
32200 DATA [illegible]
    [illegible]
    [illegible]
32300 DATA [illegible]
    [illegible]
32301 DATA [illegible]
    [illegible]
32302 DATA [illegible]
    [illegible]
32303 DATA [illegible]
    [illegible]
32304 DATA [illegible]
    [illegible]
32305 DATA [illegible]
    [illegible]
32306 DATA [illegible]
    [illegible]
32307 DATA [illegible]
    [illegible]
32308 DATA [illegible]
    [illegible]
32309 DATA [illegible]
    [illegible]
32310 DATA [illegible]
32311 UA$="[illegible]"
    :UA=INT(34*RND(U0))
    :IF UA<20 THEN UA=INT(34*RND(U0))
32312 UB=INT(34*RND(U0))
    :IF UB>7 THEN UB=INT(34*RND(U0))
32313 IF UA=U0 THEN UA=U1
32314 IF UB=U0 THEN UB=U1
32315 UB$(U1,U1)=UA$(UA,UA)
    :UB$(U2,U2)=UA$(UB,UB)
    :UB$(3,3)=UA$(UA,UA)
    :UB$(4,4)=UA$(UB,UB)
    :RETURN
32321 ? "¶~~~~|¶←~To~get~THIS~CHARACTER,
    ~Press~~~~~~~|~|¶←~CTRL~&~THIS~Key"
    :? "~~~~|~|~|¶←~INVERTED~CHARACTER"
32322 ? "~~~~¶↓~¶↓~¶↓"
    :? "~~~~[illegible]"
    :? "↓~~~~[illegible]"
32323 ? "~~~[illegible]"
    :? "↓~~~~[illegible]"
    :? "↓~~~~[illegible]"
    :? "↓~~~~[illegible]"
32324 ? "~~~[illegible]"
    :? "↓~~~~[illegible]"
    :? "↓~~~~[illegible]"
32325 ? "↓~~~¶¶=ESC,ESC~~¶▶=ESC,TAB~~[illegible]¶¶,
    SPACE~~~~~~~¶¶=ESC,SHIFT-CLEAR
~~¶◀=ESC,BACK~S"
    :? "~~~~~~¶[illegible]=ESC,CTRL-2~";
32326 ? "~~@=SHIFT-[illegible]"
    :GOSUB BTIME
    :GOTO 29985
```

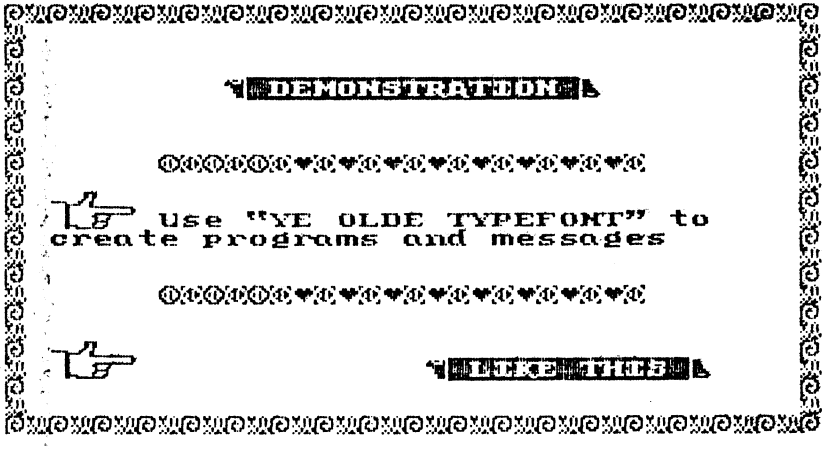
NOTES

NOTES

FANCY FONTS

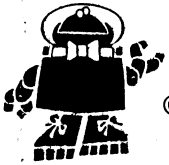
by
Richard Wilson and
Stan Gilbert

The font styles of past and future ages come alive in displays like you have never seen before. Learn to create custom fonts with the special editor included. We show you how to use these fonts in special displays on the screen using YE OLD TYPE FONT. Many extra font styles are included for you to use. Printer owners can list their fonts on paper to study and modify, and if you have an EPSON FX80, you can place your character set into it's memory and actually print everything in the style of your choice.



REQUIRES:
Basic Cartridge
16K Tape or 32K Disk

OPTIONAL:
Printer



©1983 by Educational Software Inc

Atari is a registered trademark of Atari, Inc. ESI and Tricky Tutorial are trademarks of Educational Software Inc.

OR
=
HEN
= THIS: IF THIS = 15 THEN
1) = Y THEN POKE 427,Y: Z
E 8
O X
,Ø,1